VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

PRÁCE S HISTORICKÝMI MAPAMI NA MOBILNÍM ZAŘÍZENÍ

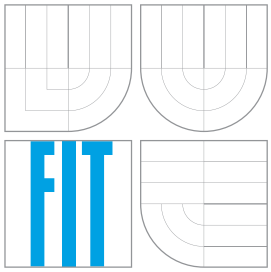
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN URBAN

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

PRÁCE S HISTORICKÝMI MAPAMI NA MOBILNÍM ZAŘÍZENÍ

INTERACTION WITH OLD MAPS ON MOBILE DEVICES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN URBAN

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. VÍTĚZSLAV BERAN, Ph.D.

BRNO 2014

Abstrakt

Cílem této diplomové práce je experimentovat s nejmodernějšími webovými technologiemi a navrhnout nový postup pro tvorbu mobilních aplikací. Díky navženým postupům je možné vytvářet mobilní aplikace, které jsou multiplatformní a zároveň téměř nerozpoznatelné od nativních. Důraz je kladený na výkon a nativní chování uživatelského rozhraní. Popsané postupy jsou demonstrované na aplikaci pro práci s historickými mapami, která je schopna v reálném čase zobrazovat mapy z historických archivů po celém světě. Testy demonstrační aplikace vykazují oproti standardním postupům tvorby webových aplikací rapidní zrychlení.

Abstract

The goal of this thesis is to experiment with the latest web technologies and to design new process for mobile application creation. It is possible to create multiplatform applications which are almost unrecognizable from native applications by proposed procedures. It is focused on performance and native behaviour of the user interface in this thesis. Described practices are demonstrated on application designed for work with historical maps, which is able to show maps from historical archives whole over world real-time. Rapid acceleration has been showed on the demonstrative application compared to standard process of creation of web applications.

Klíčová slova

mobilní aplikace, webová aplikace, historické mapy, web worker, CSS hardvérové akcelerace, HTML5, CSS3, vícevláknový Javascript, Google Closure Tools, výkonný web, multiplatformní mobilní aplikace, PhoneGap, moderní webové technologie

Keywords

mobile application, web application, historical maps, web worker, CSS hardware acceleration, HTML5, CSS3, multithreaded Javascript, Google Closure Tools, web performance, multiplatfom mobile development, PhoneGap, modern web technologies

Citace

Martin Urban: Práce s historickými mapami na mobilním zařízení, diplomová práce, Brno, FIT VUT v Brně, 2014

Práce s historickými mapami na mobilním zařízení

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Vítězslava Berana Ph.D. a uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal. Další informace mi poskytl Mgr. Peter Přidal ze společnosti Klokan Technologies.

.....
Martin Urban
31. července 2014

Poděkování

Děkuji svému vedoucímu Ing. Vítězslavu Beranovi Ph.D. za odborné vedení a podněty, které mi pri řešení tohoto projektu poskytl. Dále děkuji Mgr. Petru Přidalovi ze společnosti Klokan Technologies za odborné konzultace a pomoc v oblasti historických map a za možnost vyvíjet reálně nasaditelnou aplikaci s existující uživatelskou základnou.

Obsah

Kapitola 1

Úvod

Vzhľadom k vysokej popularite mobilných zariadení, sa väčšina desktopových aplikácií presúva práve na mobilné platformy. Napriek tomu neexistuje jednoduché riešenie pre vývoj natívnych mobilných aplikácií na viacerých platformách súčasne. Možným riešením tohto problému je tvorba tzv. hybridných aplikácií, teda webových aplikácií s prístupom k hardvéru zariadenia. Ak napríklad vyvíjaná aplikácia pracuje s dátovo náročnými obrazovými informáciami, akými sú aj historické mapy vo vysokom rozlíšení, výkon poskytovaný štandardnými webovými technológiami nestačí.

Účel tejto práce je experimentovanie s najmodernejšími webovými technológiami a návrhnutie vhodného prístupu k tvorbe mobilných aplikácií. Dôraz je kladený na výkon, platformovú nezávislosť a plynulé používateľské rozhranie, ktoré sa čo najviac podobá tomu z natívnych aplikácií.

Táto práca je členená do 7 kapitol. Kapitola 2 obsahuje prehľad existujúcich technológií pre tvorbu hybridných a webových aplikácií. Ďalej obsahuje prehľad technológií pre prácu s mapami a ukážky existujúcich mapových projektov. Ďalšie tri kapitoly tvoria jadro práce. Zaoberajú sa skúmaním jednotlivých technológií a návrhom ich použitia v kombinácií s inými technológiami. Kapitola 3 sa orientuje na problematiku komunikácie so serverom ako i medzi jednotlivými časťami aplikácie. Kapitola 4 sa zaoberá vylepšovaním výkonu a kapitola 5 je zameraná na používateľské rozhranie. Kapitola 6 je venovaná aplikácií, ktorá demonštruje navrhnuté postupy a riešenia. Návrh a realizácia aplikácie je nasledovaná podkapitolou o testovaní. Testovaný je prínos navrhnutých postupov a technológií na implementácií demonštračnej aplikácie. Záverečná kapitola 7 obsahuje zhrnutie výsledkov a smery ďalšieho vývoja demonštračnej aplikácie.

Demonštračná aplikácia slúži na prehliadanie historických máp, ktoré sú sprístupňované rôznymi svetovými i českými inštitúciami. Snaží sa ich atraktívnym spôsobom sprístupňovať širokej verejnosti. Týmto demonštračná aplikácia napĺňa jeden z pozitívnych trendov modernej doby, ktorým je záchrana kultúrneho dedičstva - napríklad aj historických dokumentov a máp.

Kapitola 2

Platformy a technológie

Multiplatformový vývoj mobilných aplikácií v súčasnosti nie je úplne vyriešený problém. Každá spoločnosť si presadzuje svoj vývojový jazyk, prostredie a podporované technológie. Napriek tomu existujú možnosti, akými tieto obmedzenia obísť. Jedným zo spôsobov ako to dosiahnuť je využitie webových technológií. Veľké množstvo aplikácií práve zobrazuje informácie prostredníctvom webového rozhrania, a teda hlavná časť aplikácie beží v rámci webového rámca. Do tejto kategórie patria napríklad mapové aplikácie.

Ak sa rozhliadneme po existujúcich frameworkoch, ktoré umožňujú tvorbu natívnych aplikácii pre rôzne platformy, nájdeme ich hneď desiatky. Zvyčajne sa jedná o kombináciu HTML5, CSS3 a javascriptu. Môže sa jednať o kompletné riešenie ale i kombináciu frameworkov. Jeden pre vlastnú implementáciu funkčnosti webovej aplikácie pomocou javascriptu, ktorý sprístupňuje API mobilného zariadenia. Druhý pre používateľské rozhranie s optimalizovaným CSS pre rôzne zariadenia.

2.1 Frameworky pre sprístupnenie API mobilného zariadenia

Pri tvorbe multiplatformových aplikácií je potrebné použiť framework ktorý zabezpečí vrstvu (*web-to-native*), ktorý vyrovná rozdiely medzi jednotlivými platformami a zjednoteným spôsobom sprístupní API zariadení prostredníctvom javascriptu. U webových aplikácii sa v podstate začlení rámec s internetovým prehliadačom do natívnej aplikácie. Existuje niekoľko frameworkov a každý má podobné vlastnosti a možnosti. Každý spĺňa to čo je pre vývoj nutne potrebné a pridáva niečo špeciálne, čím si snaží získať používateľov.

Pre bežných používateľov sú vhodnejšie kompletné riešenia, ktoré poskytujú **Intel App's framework**[1] alebo **Appcelerator Titanium**[2]. Framework od Intelu je komplexnejší a poskytuje preddefinované používateľské prvky pre jednoduchšiu tvorbu UI, ktoré napodobňujú natívne aplikácie. Je možné využívať rôzne šablóny pre rôzne platformy. Javascriptová knižnica je podmnožinou populárnej *jQuery* knižnice. Je možné použiť i plnohodnotnú jQuery knižnicu pre maximalizáciu funkcionality. Dokumentácia a kvalitné návody sú samozrejmosťou. Oproti Appcelerator Titanium nepodporuje platformu Windows Phone. Appcelerator poskytuje oproti ostatným spomínaným frameworkom cloudové predkompilované API služby. Medzi tieto služby patrí napríklad správa používateľov, zdieľanie fotografií, integrácia sociálnych sietí a mnoho ďalších.

[1] http://app-framework-software.intel.com
[2] http://www.appcelerator.com/

Pre väčšiu slobodu vo voľbe javaskriptovej knižnice a frameworku pre tvorbu používateľských rozhraní sú vhodné frameworky **PhoneGap**[3] od Adobe a **CocoonJS**[4] od Ludei. Obe frameworky sú veľmi podobné. CocoonJS je začínajúci projekt, ktorý nemá tak veľkú komunitu ako PhoneGap. CocoonJS podľa oficiálnych stránok sľubuje vyššiu rýchlosť a možnosti ako PhoneGap, avšak pre plné využitie potenciálu frameworku je potrebné napísať motivačný list autorom pre pridelenie prémiového účtu. Navyše sa nejedná o *OpenSource* a pri štarte aplikácie sa zobrazuje reklamné logo.

Ak je pri vývoji preferovaná kvalitná dokumentácia, množstvo zásuvných modulov (*pluginov*), rôzne možnosti prístupov k vývoju a to všetko zadarmo, je najvhodnejším frameworkom PhoneGap. Neobsahuje vlastné ucelené vývojové prostredie. Prekladá sa z príkazového riadku alebo pomocou cloudovej služby. Výhodou je možná integrácia cloudovej služby s *GitHub*-om[5], ktorá umožňuje tvorbu rôznych verzií aplikácie. Výhodou je bohaté spoločné API sprístupňujúce interné API mobilných platforiem prostredníctvom javascriptu a spomínaná kvalitná dokumentácia. Prí vývoji javascriptu je možné použiť ľubovoľnú knižnicu a vytvoriť si vlastné prekladové prostredie vďaka programu *make*. Framework je neustále vyvíjaný a aktualizovaný.

Pri vzdialení sa od webových aplikácii stojí za zmienku **Corona SDK**[6]. Vývojovým jazykom je *C++* s podporou *OpenGL*. Podporované platformy sú Android a iOS. Tento framework je vhodný na tvorbu hier a aplikácií, umožňuje použitie natívnych Objective-C a Java knižníc. Corona SDK je platený framework. Existuje i Starter verzia ale je značne obmedzená[7].

2.2 Frameworky pre tvorbu webových používateľských rozhraní

Po voľbe frameworku pre zabalenie webovej aplikácie do natívnej, prichádza na rad voľba nástroja pre tvorbu používateľského rozhrania. Variant je opäť niekoľko a ich použitie je na rôznych úrovniach tvorby rozhrania.

Je možné zvoliť kompletné riešenie, kde je obrovské množstvo preddefinovaných používateľských prvkov, podporujé rôzne témy pre jednotlivé platformy. Z týchto veľkých systémov je možné spomenúť **jQuery Mobile**[8] alebo skôr jeho súčasť **jQuery UI**. Sú založené na známej *jQuery* knižnici pre javascript, a stali sa pevným základom i pri tvorbe mobilných aplikácií. Slúži pre tvorbu webových stránok a aplikácií s optimalizáciou na rôzne dotykové platformy. Výhodou je databáza plná preddefinovaných elementov. Navyše podporuje témy, čím je možné napríklad nastaviť odlišné štýly pre rôzne platformy. Ďalším z robustných systémov je **Sencha Touch**. Veľmi obľúbený framework HTML5 s MVC (*model-view-control*) prístupom pre tvorbu mobilných aplikácií. V použití s *PhoneGap* frameworkom vzniká silný nástroj na tvorbu kvalitných aplikácií. Obsahuje množstvo vstavaných komponentov, plynulé animácie, rolovanie i prispôsobivé rozloženie komponent na rôznych obrazovkách. Spolu s ďalšími Sencha nástrojmi vzniká sada nástrojov, ktorá obsahuje všetko, čo je pre tvorbu aplikácie potrebné. Základný framework Sencha Touch je zadarmo. Ostatné nástroje a podpora je však platená. Kompletný balík sa pohybuje na cene okolo 1000 dolárov.

[3] http://www.phonegap.com
[4] https://www.ludei.com/cocoonjs/
[5] http://github.com/
[6] http://www.coronalabs.com/
[7] http://coronalabs.com/products/corona-sdk/starter/
[8] http://jquerymobile.com/

Framework	Programovací jazyk	Kompletné vývojové prostredie (XDK)	Podpora pluginov	Platformy	Cloudová služba	Reklama v aplikácií	Open Source	Cena
Adobe PhoneGap	HTML/ CSS3/ Javascript	nie	áno	iOS, Android, Windows Phone, Blackberry, Symbian, Bada, WebOS	Preklad, 1 privátna aplikácia zadarmo, počet verejných aplikácií obmedzený nie je.	nie	áno	Zadarmo
CocoonJS by Ludei	HTML/ CSS3/ Javascript	nie	áno	iOS, Android, Windows Phone, Blackberry, Tizen, Facebook, NOOK, a pod.	Preklad, pre pohodlné používanie potrebný Premium účet.	áno	nie	Zadarmo
Intel App's framework	HTML/ CSS3/ jQuery	áno	áno	Android, iOS, Blackberry, Windows 8	nie	nie	áno	Zadarmo
Appcelerator (Titanium)	HTML/ CSS3/ Javascript	áno	nie	iOS, Android, Windows Phone, Blackberry	Len verejné API služby (obmedzene) .	nie	SDK áno	Verzia Developer zadarmo
Corona SDK	C++/ OpenGL	nie	áno (Pro)	iOS, Android, Kindle Fire, NOOK	nie	nie	nie	Basic 16€/mes. Pro 49€/mes.

Obrázek 2.1: Porovnanie frameworkov pre sprístupnenie API mobilných zariadení

Ak je potrebné aplikáciu iba rýchlo vytvoriť prototyp aplikácie, je možné použiť nástroje ako Ratchet alebo TopCoat. **Ratchet** je vytvorený na prototypovanie iOS a Androidu a taktiež obsahuje iba tieto 2 témy. Adobe **TopCoat**[9] je pokročilejší framework, ktorý je implementovaný iba prostredníctvom HTML a CSS3. Je optimalizovaný na maximálny výkon, čistý kód a jednoduchú tvorbu štýlov jednotlivých elementov. Tiež poskytuje jednoduchú zmenu dizajnu prvkov - tvorbu tém pomocou CSS.

Posledná kategória sa týka jednoduchých malých frameworkov ktoré sú zamerané na konkrétnu časť používateľského rozhrania. Príkladmi môžu byť SideTap alebo Snap(SnapJS). **SideTap**[10] je zameraný na jednoduché riešenie prechodov jednotlivých obrazoviek. Vyžaduje jQuery knižnicu. **SnapJS**[11] je zameraný na tvorbu animovaných, vysúvacích, bočných menu, moderných naprieč všetkými platformami. Je platformovo nezávislý a kompatibilný s *Ratchet* frameworkom.

Framework	Účel frameworku	Zameranie na platformy	Témy	Jazyky
jQuery UI (jQuery Mobile)	Tvorba uživatelských rozhraní v javascripte prostredníctvom jQuery, vrátane preddefinovaných používateľských prvkov. Súčasť jQuery Mobile,	univerzálne	áno	HTML/CSS/ jQuery
Ratchet	Prototypovanie aplikácií.	iOS/Android	iOS a Android	HTML/CSS/JS
TopCoat	Elementy dizajnované pomocou CSS3 a optimalizované pre maximálny výkon. Vhodné na implementáciu UI.	univerzálne	áno	HTML/CSS
SideTap	Tvorba uživaťelských rozhraní pomocou vysúvacích obrazoviek.	univerzálne	áno, vlastné	HTML/CSS/ jQuery
Snap	Tvorba posuvných gest. Prechody medzi obrazovkami, vyťahovacie menu.	univerzálne	nie	Javascript
Sencha Touch	MVC systém s velkou sadou komponent a tém.	univerzálne	áno	HTML/CSS/JS

Obrázek 2.2: Porovnanie frameworkov pre tvorbu používateľských rozhraní mobilných zariadení

2.3 Google Closure Tools

Pre vytvorenie funkčnosti webovej aplikácie je nevyhnutný javascript. Čistý javascript je ťažko použiteľný v dôsledku zložitých konštrukcií a rôznej interpretácií v jednotlivých prehliadačoch. Preto je vhodné použiť niektorou z dostupných knižníc. Jednou alternatívou pri mobilných aplikáciach je *jQuery Mobile*, ktorá bola spomenutá v predchádzajúcej sekcii. Táto podkapitola sa zameriava na *Google Closure*. Jedná o javascriptovú knižnicu, ktorá je veľmi vhodná pre tvorbu mobilných webových aplikácií. Výhody Google Closure[12] sú v komplexnosti knižnice. Poskytuje množstvo nástrojov, ktoré je možné, ale nie nevyhnutné, používať. Bola vyvinutá ako interná knižnica pre tvorbu google webových aplikácií ako *Google mail*, *Google mapy*, *Google kalendár* a podobne. Kód písaný v tejto knižnici

[9] http://topcoat.io/
[10] https://github.com/harvesthq/sidetap
[11] https://github.com/jakiestfu/Snap.js/
[12] https://developers.google.com/closure/?hl=cs

je následne skompilovaný na čistý a optimalizovaný javascript s maximálnou výkonnosťou. Podľa [8] je potrebné, pre dosiahnutie lepšieho výkonu, dodržiavať množstvo konvencii pri písaní javascriptu. Pri použití Google Closure to za vývojára vykoná prekladač (*kompilátor*). Webové aplikácie, vytvorené prostredníctvom Google Closure Tools, sú platformovo nezávislé. Mobilné platformy nie sú výnimkou. Do výsledného produktu nie je potrebné prikladať žiadne ďalšie súbory. Skompilovaný výstup je čistý a minimalistický javascript. Voliteľne je možné prepínať úrovne optimalizácií, možnosti ladenia ako i zapnúť obfuskáciu[13] kódu. Google Closure Tools umožňuje i tvorbu šablón prostredníctvom *Google Closure Templates*. Táto časť práce je založená na [4].

Šablóny

Google Closure Templates je nástroj na tvorbu šablón webových aplikácií. Je použiteľný ako v v jazyku Java tak i v Javascripte. Cestou šablón je jednoduché meniť obsah aplikačnej stránky a zároveň vždy zabezpečiť správne zobrazenie a jednoduchú modifikáciu všetkých ekvivalentných stránok. Rovnako umožňuje načítanie javascriptu na pozadí stránky. Šablóny sa tvoria ako dokumenty typu *SOY*. V takomto dokumente sa vytvorí ľubovoľné množstvo šablón, ktoré majú parametrizovateľný obsah. Primárne použitie je v rámci generovania HTML alebo CSS, ale v konečnom dôsledku i akéhokoľvek iného textu. Pri kompilácií vzniká zo šablóny javascriptová funkcia.

Javascript

Closure tools nám umožňuje písať typovaný javascript s podporou modularity. Prekladač pri kompilácií eliminuje mŕtvy kód, kontroluje prípadné chyby a celý kód optimalizuje. Samozrejmosťou je skracovanie názvov, predbežné priraďovanie do premenných mimo cyklov, rušenie medzier a odsadenia. Výsledný kód je po vypnutí ladenia nečitateľný. Naopak v rámci zdrojových súborov sa kladie dôraz na komentáre funkcií, parametrov i premenných. Pri správnom nastavení prekladača, nie je možne preložiť nekomentovaný kód. Ďalším nástrojom z Closure Tools je *Linter*, ktorý dokáže kontrolovať a čiastočne opravovať chyby v programátorskom štýle (odsadenie, dĺžka riadkov a pod.). Medzi veľké prednosti patrí podpora pre navigáciu medzi stránkami bez potreby obnovenia stránky, jednoduchú prácu s *DOM* objektami, *iFrame* vnorenými rámcami a animáciami, ako i množstvo preddefinovaných objektov. Napriek tomu, že je knižnica veľmi robustná, vôbec to nemá vplyv na veľkosť alebo rýchlosť aplikácie. Dôvodom je kompilácia.

2.4 Kartografické dáta a mobilné zariadenia

Táto podkapitola sa zaoberá webovými technológiami v oblasti kartografie v kontexte mobilných zariadení. Mobilné zariadenia otvárajú nové možnosti pre hľadanie, zobrazovanie a interakciu s veľkým množstvom oskenovaných a georeferencovaných[14] dát. V texte sú popísané možnosti získavania historických máp, technológie umožňujúce vyhľadávať tieto mapy a spôsoby akými sú digitalizované mapy umiestňované na geografickú polohu.

[13] obfuskácia je odvodené z anglického slova *obfuscate* - zatemniť

[14] Georeferencing/Georeferencovanie - zasadenie rastru do súradnicového systému

Získavanie historických máp

Vďaka projektom spoločnosti *Klokan Technologies* v spolupráci s partnermi ako sú *The British Library*, *National Library of Scotland* a desiatok ďalších inštitúcií i z Českej republiky vznikli projekty OldMapsOnline.org a StareMapy.cz. Z tohto dôvodu je teraz k dispozícií veľké množstvo oskenovaných máp, ktoré vďaka *crowdsourcingu*[15] a úsiliu dobrovoľníkov boli umiestnené na ich geografickú polohu. Počet týchto máp neustále rastie. Tieto mapy je možné zobrazovať a atraktívnym spôsobom ich sprístupňovať prostredníctvom webových technológií.

Vyhľadávanie v historických mapách

Servery spomínaných inštitúcií obsahujú veľké množstvá máp[16].

Technológia ktorá v rámci týchto serverov dokáže vyhľadávať sa nazýva **MapRank Search**[17]. Jedná sa o technológiu umožňujúcu vizuálne vyhľadávanie máp prostredníctvom vymedzenia hľadaného územia. Táto technológia prináša možnosť vyhľadávania podľa oblasti vymedzenej v geografických súradniciach, intervalov doby vzniku mapy, mierky mapy a textu zo záznamov obsahujúcich metadáta. Dôraz je kladený na rýchlosť odozvy. Používateľ i pri milióne záznamov dostane najrelevantnejšie výsledky v čo najkratšej dobe a tak je možné vyhľadávať a aktualizovať záznamy vo veľmi krátkom čase[18].

Výsledky vyhľadávania sú prostredníctvom odkazu prepojené späť do katalógu, kde sa nachádzajú úplné informácie o mape a ďalšie podrobnosti. Systém je možné integrovať do webových stránok jednotlivých knižníc, ktoré mapy do katalógu poskytli, avšak s obmedzením len na tieto mapy.

Georeferencovanie máp

Knižnice, archívy, múzea a inštitúcie kultúrnych dedičstiev masívne začali digitalizovať svoje zbierky a publikovať ich. Oskenovaná mapa je ale len obrázok a nie mapa ako ju poznáme v rámci informačných technológií. Nie je možné nájsť polohu mapy vo svete ani pozíciu na mape napr. prostredníctvom *GPS* (globálny pozičný systém). Nie je možné ani porovnávať mapy medzi sebou. V tomto dôsledku je nutné mapu umiestniť na jej geografickú polohu a tým ju premeniť na hodnotnejšiu mapu s možnosťou väčšej interakcie.

Existuje niekoľko nástrojov na georeferencovanie máp. Príkladom môže byť používateľsky prívetivý open-source geografický informačný systém pre tvorbu vlastných geografických projektov. Obsahuje i prídavný modul na georeferencovanie dát. Jeho názov je **QGIS Georeferencer Plugin**[19].

Ďalší nástroj na tvorbu vzťahov medzi obrázkom a geografickou polohou vyvinula spoločnosť PocketDLog. Jedná sa o **Delta Data Systems Georeferencer**[20], ktorý pracuje

[15] Crowdsourcing - novotvar na označenie spôsobu deľby práce, pri ktorom sa úloha (obvykle vykonávaná zamestnancom alebo konktraktorom) zadá bližšie nešpecifikovanej skupine ľudí ako všeobecná výzva. Napríklad sa verejnosť vyzve na spoluprácu pri vývoji novej technológie, uskutočnení dizajnérskej úlohy, zdokonalení algoritmu alebo pri pomoci so zachytením, roztriedením a analýzou veľkého objemu dát.

[16] Iba v rámci zbierky Karlovej Univerzity v Prahe je k dispozícii okolo 130 tisíc historických máp. Bližšie informácie o množstve máp jednotlivých inštitúcií je možné nájsť na adrese http://project.oldmapsonline.org/collections.

[17] http://www.mapranksearch.cz/

[18] Pod pojmom krátky čas sa myslí možnosť použitia technológie v aplikácií, kde sú mapy vyhľadávané v reálnom čase ako odpoveď na zmeny používateľského rozhrania.

[19] http://qgis.org

[20] http://www.deltadatasystems.com/prod4.htm

s GPS a umožňuje navigáciu po týchto georeferencovaných mapách.

Javascriptový **Georeferencer**[21] od Klokan Technologies je nástroj, ktorý umožňuje prevod oskenovanej mapy na skutočnú elektronickú mapu, ktorá môže byť jednoducho integrovaná na webových stránkach. Ďalej umožňuje atraktívnu prácu s mapami, 3D vizualizáciu pomocou Google Earth. Dokáže prekrývať mapy, poskytovať štandardné mapové služby a vyhľadávať v mapách na základe MapRank-u[22]. Vďaka tomu, že sa jedná o webovú službu, je možné tento nástroj využívať i vo webových aplikáciach.

Zobrazovanie máp vo webových aplikáciach

V rámci webových technológií existuje niekoľko rôznych prehliadačov máp. Táto sekcia je zameraná na *Google Maps* a *Open Layers*. Podľa správy od organizácie GlobalWebIndex[23] sú **Google mapy**[24] najpoužívanejšou aplikáciou na chytrých mobilných telefónoch za obdobie druhého kvartálu 2013 (apríl, máj, jún), čo reflektuje kvalitu a výkon máp. Navyše Google poskytuje webové API, pre použitie máp integrovaných do webových aplikácií a v dôsledku toho i do mobilnej webovej aplikácie. Mapový prehliadač od Googlu ale nie je vhodný na zobrazovanie historických máp, lebo mu chýba priama podpora pre zobrazovanie vlastných mapových dát. Pre túto úlohu je vhodnejší mapová prehliadač **Open Layers 3**[25], ktorý napriek tomu, že sa v čase písania tohto textu sa nachádza v ešte nefinálnej verzii, je komplexná a výkonná knižnica pre prácu s mapami. Medzi jej najväčšie výhody patrí podpora mobilných zariadení, viacvrstvové mapy, jednoduchá prispôsobiteľnosť, možnosť použiť mapové dlaždice z MapBox, Bing či OSM. Jedná sa o Open source projekt.

Vhodnou aplikáciou pre online publikovanie máp je i **MapTiler**[26] od spoločnosti Klokan Technologies. Vďaka aplikácií je možné tvoriť prekrývajúce sa mapy napríklad nad podkladovými Google mapami a inými. Základom je práca s mapami vo vysokom rozlíšení, ktoré sú predom vyrenderované a rozdelené na dlaždice a tým umožňujú rapídne navýšenie výkonu pri zobrazovaní a práci s mapami. Súčasťou je prehliadač TileServer[27], ktorá dokáže dlaždice máp zo serveru zobrazovať pomocou rôznych mapových prehliadačov, vo viacerých vrstvách a priamo v internetovom prehliadači.

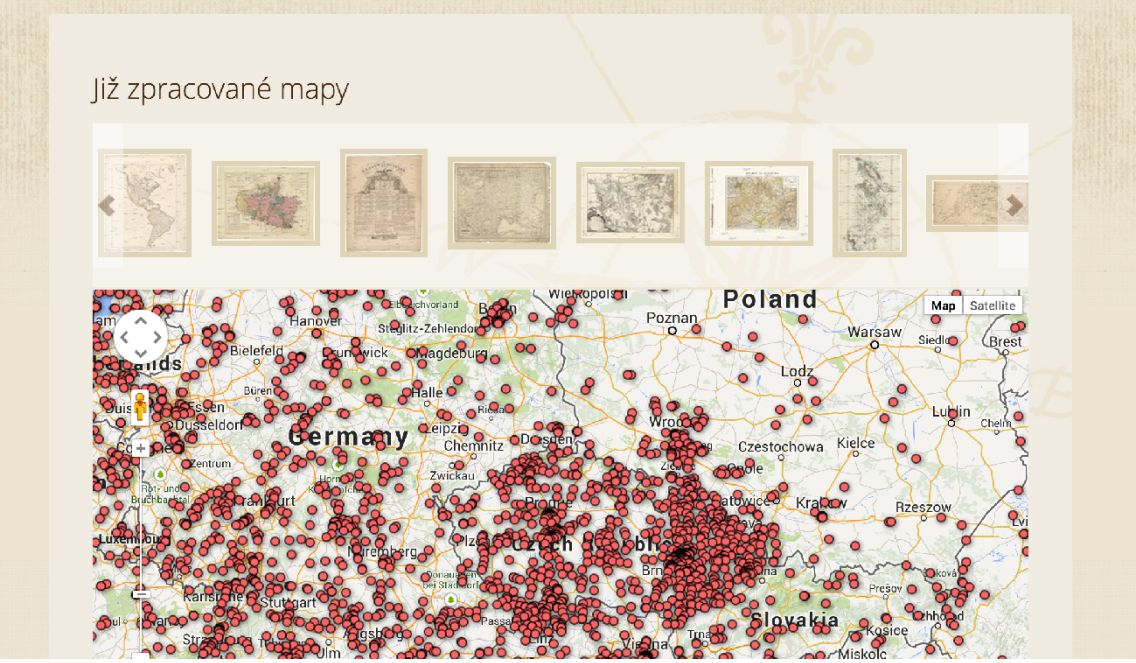
Analýza súčasných riešení pre prácu s historickými mapami

V súčasnosti je možné pristupovať k historickým mapám cez webový prehliadač. Pri použití na mobilnom zariadení je najväčšou nevýhodou neprispôsobené používateľské rozhranie, problémy s približovaním na niektorých platformách a nekompatibility s rôznymi internetovými prehliadačmi. Navyše sa pri tejto variante nevyužíva potenciál mobilného zariadenia.

Jedným z prvých projektov so zameraním na historické mapy a dokumenty je **Staré a historické mapy - Sbírky starých map v České Republice**[28]. Ukážku webových stránok je vidieť na obrázkoch 2.3 a 2.4. Projekt sa týka prevažne Českej republiky. Má za cieľ obnoviť, zachovať a sprístupniť staré mapy pre ďalšie generácie a vedecké inštitúcie formou súťaže. Jedným z hlavných partnerov je Moravská zemská knihovna. Ďalšími partnermi sú

[21] http://www.klokantech.com/georeferencer/
[22] Viac v sekcii MapRank Search kapitoly 2.4 na strane 9
[23] https://www.globalwebindex.net/
[24] http://maps.google.com/
[25] http://ol3js.org
[26] http://www.maptiler.org/
[27] http://tileserver.maptiler.com/
[28] www.staremapy.cz

Masarykova Univerzita v Brne a Karlova univerzita v Prahe. Mapy sa dajú na stránke spracovávať, georeferencovať, ale aj prehliadať. Je možné ich zobraziť nad existujúcimi *Google Mapami* alebo prostredníctvom *Google Earth* do trojdimenzionálneho zobrazenia.



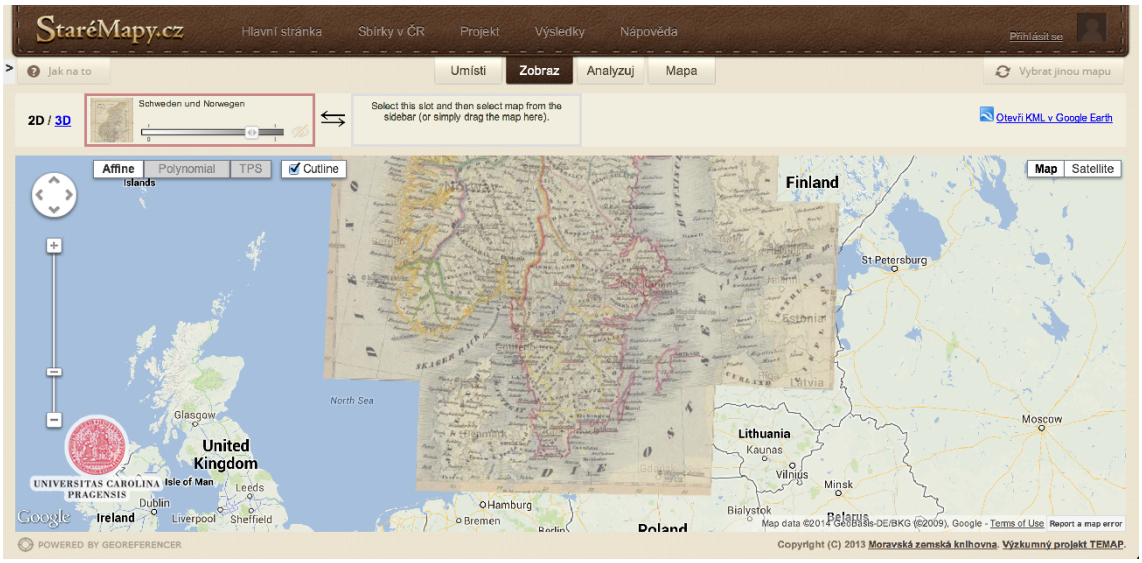
Obrázek 2.3: Ukážka webových stránok projektu Staré mapy - vyhľadávanie máp

Old Maps Online[29] je podobný projekt, ktorý naväzuje na projekt *Staré a historické mapy - Sbírky starých map v České Republice*. Jeho cieľom je zdokonaliť vyvinuté nástroje a použiť ich na celosvetovú zbierku máp. Umožňuje primárne prehliadať staré mapy s ohľadom na zobrazenú časť súčasnej mapy. Historické mapy sú vyhľadávané a zoradené podľa zodpovedajúcej viditeľnej časti mapy so zobrazeným výrezom sveta. Vyhľadávanie je filtrované na základe časovej osi - časového rozsahu a zobrazenej oblasti mapy. Ukážka je na obrázku 2.5.

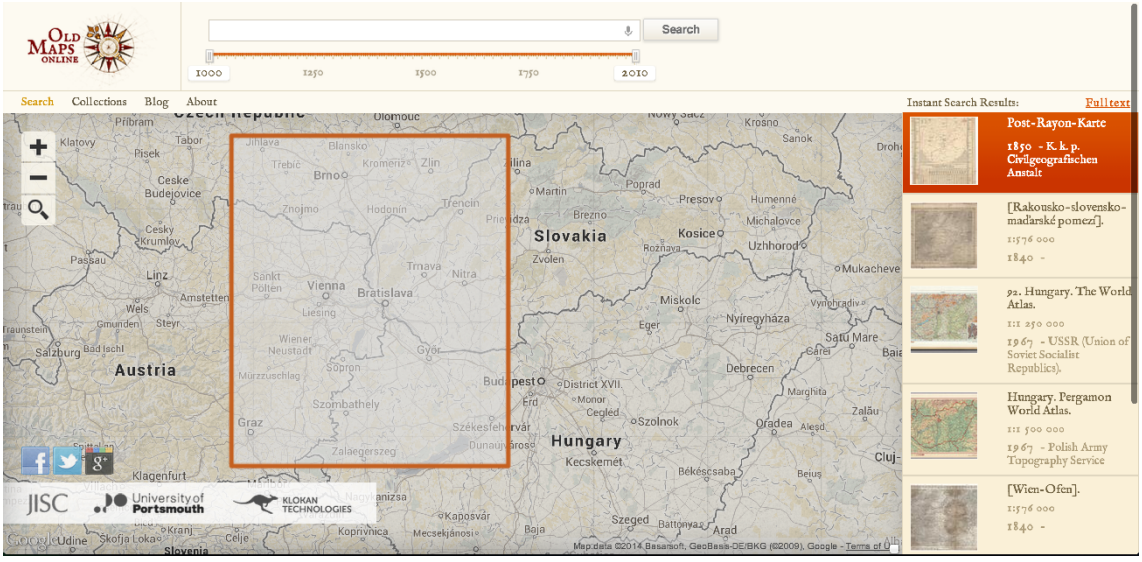
Ďalší ekvivalentný projekt, avšak zameraný na švajčiarske inštitúcie je **Kartenportal.CH**[30] (Obrázok 2.6). Výhodou sú lepšie parametre pri vyhľadávaní a filtrovaní konkrétnych máp:

- vymedzenie časového rozsahu, z ktorého obdobia mapa pochádza
- miesta (región, mesto)
- atribútov (obsiahnutý text v popise, názov, autor, vydavateľstvo, mierka mapy a pod.)
- názvu konkrétnej inštitúcie

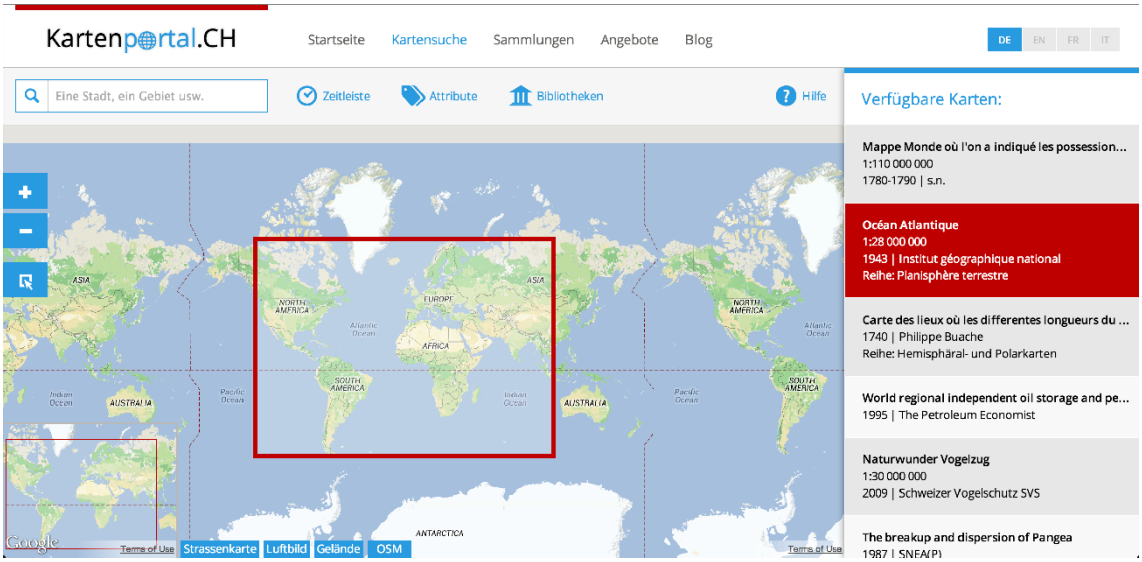
[29] www.oldmapsonline.org
[30] www.kartenportal.ch



Obrázek 2.4: Ukážka webových stránok projektu Staré mapy - detail konkrétnej mapy



Obrázek 2.5: Ukážka webových stránok projektu Old Maps Online



Obrázek 2.6: Ukážka webových stránok projektu Kartenportal.CH

Kapitola 3

Komunikačné kanály

Jednotlivé bloky aplikácie potrebujú medzi sebou komunikovať. Napríklad medzi sebou komunikujú jednotlivé obrazovky, používateľské rozhranie so zadnou časťou aplikácie. Pri klient-server aplikáciach komunikuje i celá klientská aplikácia so serverovou časťou.

Táto kapitola sa v prvej podkapitole bude zaoberať práve získavaním dát zo serveru. Ďalšou časťou bude riešenie komunikácie medzi obrazovkami aplikácie, respektíve obrazovkami vnorenými rámcami iFrame na aplikačnej úrovni. Poslednou podkapitolou bude napojenie webovej aplikácie na aplikačné rozhranie mobilného zariadenia.

3.1 Získavanie dát zo serveru

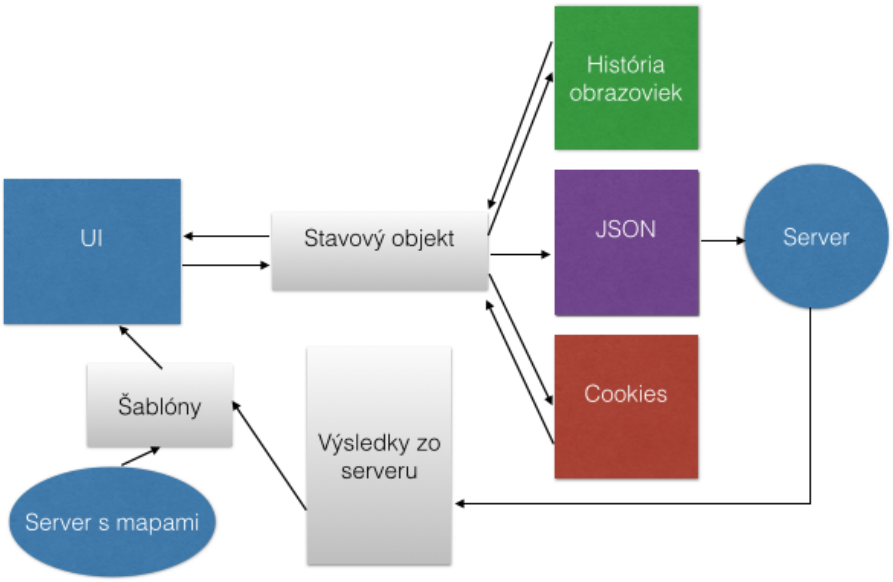
Najdôležitejšou a najzložitejšou časťou projektu je často získavanie dát zo serveru a práca s nimi. Aplikáciu obrovské množstvo sťahovaných dát spomaluje a tento proces navyše energeticky vyťažuje zariadenie a zahlcuje dátovú linku. Je nutné zabezpečiť aby sa sťahovalo iba to, čo je naozaj potrebné a vtedy, keď je to potrebné. Ideálne mimo vedomia uživateľa.

Existujúce riešenia: V bežnej praxi sa na takéto problémy využíva komunikácia prostredníctvom asynchrónnych dotazov na server pomocou *XMLHttpRequest*(ďalej ako *XHR*) využívané v technológií *AJAX*[1]. XHR dotazy slúžia na zmenu obsahu webovej stránky bez potreby znovunačítania. Ďalšia technológia nazývaná *JSONP* (json-in-padding) umožňuje, na rozdiel od AJAX-u, získavať dáta i zo vzdialeného serveru (cross-origin). V najmodernejších prehliadačoch (i tých používaných na mobilných zariadeniach) je možné pre AJAX nastaviť tzv. *CORS* hlavičky (*Cross-Origin Request*)[2] a môžme posielať AJAX dotazy i na vzdialené servery.

Celý proces navrhnutého riešenia najlepšie popíše obr. 3.1. Uživatelské rozhranie zobrazuje a kontroluje aktuálny stav aplikácie. Stav je udržiavaný v rámci kódu v stavovom objekte - ten je podľa potreby konvertovaný na URL „hash“ alebo JSON. „Hash“ je časť adresy za znakom mriežky(#). Slúži ako prostriedok komunikácie medzi iFramami a je bližšie popísaný v kapitole 3.3. *JSON* je formát, ktorý slúži ako formát pre prenos dát medzi klientom a serverom. Aktuálny stavový objekt bude vždy prevedený na JSON, ktorý slúži na parametrizáciu dotazov pre server. Server následne vráti relevantné výsledky zodpovedajúce parametrom dotazu. Tieto výsledky budú mať formát JSON objektu. Tento objekt bude slúžiť ako parameter pre tvorbu výslednej stránky. Tá je vygenerovaná prostredníctvom predpripravenej šablóny dosadením parametrov. Navrhujem použiť radšej AJAX-ové rieše-

[1] http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications
[2] http://techblog.constantcontact.com/software-development/using-cors-for-cross-domain-ajax-requests/

nie s nastavenými CORS hlavičkami ako JSONP. Dôvodom sú lepšie možností využitia *web workerov*, ktoré sú popísané v kapitole 4.2.



Obrázek 3.1: Návrh získavania dát a ich zobrazovania

Servery sú pripravené v rámci spomínaných projektov pre prácu s mapami v kapitole 2.4 (str. 10). V demonštračnej aplikácií navrhnutú schému pre prácu so serverom dodržujem. Nedošlo akurát k využitiu *cookies*, nakoľko to demonštračná aplikácia nevyžaduje. Aplikácia pracuje s JSON dátami - textovými. K sťahovaniu náročných dát - máp - dochádza až pri zobrazení.

3.2 Získavanie máp zo serveru

V moderných mobilných aplikáciach zameraných na prácu s mapami nie sú vo webovom rozhraní k dispozícií natívne mapy (napr. z iOS) a už vôbec nie rovnaké na všetkých platformách. Aké mapy teda použiť? Ako tieto mapy získavať a zobrazovať?

Jednotlivé webové mapové služby a prehliadače sú popísané v kapitole 2.4 na strane 10. Webové služby poskytujú dlaždice máp, ktoré je možné pomocou týchto prehliadačov zobrazovať.

Z výkonového hľadiska navrhujem na mobilných zariadeniach použiť OpenLayers 3. A to i napriek tomu, že v čase písania tohto textu nebola vydaná finálna verzia. Dôvodom je jednoduchá práca s rôznymi vrstvami, jednoduchá výmena máp bez potreby vytvárať nový objekt mapy. Vývoj zameraný na podporu mobilných dotykových rozhraní. Taktiež preto že sa jedná o Open Source projekt, ktorý je možne v rámci projektu upravovať vlastným potrebám. Javascript obsluhujúci mapu je často veľmi rozsiahly, preto je potrebné si dať pozor aby bol načítaný iba raz. Vytváranie mapového objektu je potrebné taktiež zredukovať na minimálny počet. Objekt je pri zmene mapy lepšie zrecyklovať a iba vymeniť jeho jednotlivé vrstvy za nové.

Ako prehliadač jednotlivých starých máp je nutné používať OpenLayers 3, nakoľko digitalizované historické mapy sú pripravené pre túto technológiu. Ako vyhľadávacia a podkladová mapa mala slúžiť Google mapa. Avšak pre lepší výkon bola nakoniec použitá MapBox

mapa v OpenLayers 3 prehliadačke. Využitie rovnakých máp v celej aplikácií je čistejšie a nie je potrebné načítavať 2 rozsiahle javascriptové knižnice.

3.3 iFramy a komunikácia medzi nimi

Celá webová aplikácia môže byť veľmi mohutná. S tým nám vznikne mohutný *DOM* strom. Ak je zložitý, jeho prehľadávanie trvá dlhú dobu. Vo funkčnosti jednej obrazovky aplikácie sú väčšinou elementy ostatných nepoužívané. Týmto dochádza k spomaleniu aplikácie, ktoré rastie s počtom obrazoviek a elementov. Riešením môže byť použitie vnorených rámcov iFrame (ďalej ako rámec alebo iFrame).

Existujúce riešenia: Táto kapitola je inšpirovaná [1]. Postavená je na experimentoch s iFrame-ami v rámci demonštračnej aplikácie aplikácie a v rámci *môjho demonštračného príkladu*[3]. Riešenie je inšpirované implementáciou histórie v Google Closure knižnici[4]. Kde sa využíva osobitný iframe na udržiavanie histórie zmien URL adresy.

Prvým cieľom návrhu je rozdeliť strom na menšie časti. To je možné vytvorením viacerých DOM stromov. Navrhujem použitie vnorených rámcov pre jednotlivé obrazovky mobilnej aplikácie. Prináša to hneď niekoľko výhod. Ako som už spomínal, pre lepší výkon aplikácie je potrebné držať DOM strom tak ľahký ako to ide. Použitie vnorených rámcov tento DOM strom rozdeľuje na viacero menších DOM stromov v rôznych iFrame rámcoch. S tým prichádza do aplikácie možnosť udržiavať jednotlivé obrazovky nezávislé na druhých, čo sa týka javascriptu, CSS i HTML. Zároveň sú jednotlivé obrazovky izolované od ostatných a ľahšie udržovatelné.

Používanie vnorených rámcov však prináša i niekoľko problémov. Rámce iFrame, teda obrazovky aplikácie, musia medzi sebou komunikovať. Potrebujú reagovať na vzájomné udalosti a byť umiestnené v rámci obrazovky zariadenia, štýlované, potrebujú medzi sebou zdielať javascriptový kód. V následujúcom texte sú popísané možnosti a výhody rôznych typov komunikácie medzi aplikáciami.

Hash komunikácia

V rámci vnorených rámcov iFrame je k dispozícii objekt „location“, ktorý umožňuje prístup k URL. Vlastnosť *hash* objektu *location* sprístupní *hash token*, ktorý je možné čítať i meniť. Tento druh komunikácie je vhodný v prípadoch, kedy nepracuje aplikácia s rozsiahlimi dátami, ktoré je potrebné uchovávať. Jedná sa o veľmi čistý a jednoduchý spôsob ako komunikovať medzi rámcami navzájom i medzi rámcami a rodičovskými objektami. Príklad zmeny *hash tokenu* z rámca iFrame:

```
1 window.top.location.hash = 'hash';
```

Získavanie najvrchnejšieho okna v hierarchií prehliadača (*window.top*)[5] je potrebné použiť v prípade iFrame rámcov.

Na zmenu hash môžeme v javascripte reagovať pomocou *listener*-ov, ktoré čakajú na udalosti prvkov alebo objektov. Knižnica *Google Closure* prináša ešte pohodlnejšiu variantu, fungujúcu v takmer všetkých v súčasnosti najviac používaných webových prehliadačoch. Jedná sa o *goog.History* [4].

[3] https://github.com/klokan/closure-library-plovr-hello-world-skeletons/tree/master/iframes. Bol vytvorený pre testovanie iFrame komunikácie a zverejnený v rámci ukážkových aplikácií pre použitie *Google Closure + Plovr* na *GitHub* účte spoločnosti Klokan Technologies

[4] http://docs.closure-library.googlecode.com/git/class_goog_History.html

[5] http://www.w3schools.com/jsref/prop_win_top.asp

Vytvorenie inštancie objektu *goog.History* slúži pre sledovanie stavu adresového pola, na nastavenie reakcie na udalosť zmeny adresy a zapnutie sledovania zmeny *hash tokenu*[6] adresy.

```
var h = new goog.History();
goog.events.listen(h, goog.history.EventType.NAVIGATE, navCallback);
h.setEnabled(true);
```

Akákoľvek zmena *hash tokenu* zavolá funkciu navCallback, ktorá môže vyzerať napríklad takto:

```
function navCallback(e) {
  alert('Zmena stavu na "' + e.token + '"');
}
```

Zmenu *hash tokenu* je možne zabezpečiť manuálne nad goog.History objektom:

```
h.setToken('foo');
```

Príklad použitia je vytvorenie *goog.History* objektu v rodičovskom prvku, kde sa nachádzajú rámce iFrame. V rámcoch, ktoré vyžadujú interakciu s rodičom alebo iným rámcom je možné nastavovať *hash token* na unikátnu hodnotu. Spúšťacou udalosťou môže byť kliknutie na tlačítko alebo dotykové gesto. Ostatné rámce túto zmenu URL zachytia a môžu vykonať príslušnú úlohu.

V demonštračnej aplikácií na tomto princípe pracuje napríklad prepínanie obrazoviek. Ak sa po nejakej akcii v UI má zmeniť obrazovka, tak sa nastaví *hash token* na identifikátor nasledujúcej obrazovky. Rodičovský javascript túto zmenu zachytí, skryje nepotrebnú obrazovku a zobrazí požadovanú. Týmto ide dosiahnuť kompletnú navigáciu medzi obrazovkami pomocou jediného stavového automatu.

Systémová komunikácia

Predchádzajúce riešenie komunikácie iFrame rámcov nerieši prístup k dátam mimo rámcov.

Riešenie: V predchádzajúcej kapitole 3.3 bola spomenutá vlastnosť „top“ alebo „windows.top“ ako prístup k najnadradenejšiemu oknu. Tým pádom je zrejmé, že dostať sa k dokumentu rodičovského elementu, a teda i k jeho *DOM* stromu je jednoduché:

```
top.document
```

Z dokumentu rodiča je možné získať iFrame prvok (*element*) napríklad pomocou jeho identifikátoru:

```
top.document.getElementById('iframeid');
```

Knižnica *Google Closure* umožňuje získať z daného iFrame rámca príslušné okno alebo dokument:

```
goog.dom.getFrameContentWindow(top.document.getElementById('iframeid'))
goog.dom.getFrameContentDocument(top.document.getElementById('iframeid'))
```

Čistým javascriptom to ide podobne:

```
top.document.getElementById('iframeid').contentWindow
top.document.getElementById('iframeid').contentDocument // Problemy v~IE8
```

Týmto je možné sprístupniť *DOM strom* dokumentu ľubovoľného rámca. Niekedy požadujeme volať javascriptové funkcie alebo získavať objekty. Javascriptové globálne objekty sú uložené v príslušnom okne *window* resp. *contentWindow*:

[6] Pojmom *hash token* označujem časť URL adresy za znakom mriežky (#)

```
goog.dom.getFrameContentWindow(top.document.getElementById('iframeid'))['objekt']
```

Funkcia je taktiež objekt, takže volanie funkcie vyzerá následovne:

```
goog.dom.getFrameContentWindow(top.document.getElementById('iframeid'))['funkcia'](par1, par2);
```

Týmito prístupmi sa vyrieši akákoľvek javascriptová komunikácia v rámci iFrame rámcov veľmi jednoduchou, zrejmou a logickou cestou. So získanými objektami je možné pracovať rovnako ako vo vlastnom rámci.

V popise sa neuvažovalo zanorovanie rámcov do seba. Avšak tento princíp ide uplatniť na ľubovolný počet úrovní vnorených rámcov.

Jeden z návrhov môže byť preposielanie v rámci hierarchií. Pod pojmom hierarchia rámcov je možné si predstaviť strom. S pojmom komunikácia medzi iFrame rámcami sa zmení strom na cyklický graf. Jeho uzlami sú rámce. Uzly vždy komunikujú s koreňovým prvkom. Prostredníctvom koreňového prvku sa snažia volať funkcie nad priamym potomkom. Pre zabezpečenie komunikácie cez viacero úrovní, každý potomok je schopný volanie funkcie delegovať na svojich vlastných potomkov.

V demonštračnej aplikácií sa táto komunikácia využíva k informovaniu iFrame rámca zaoberajúceho sa získavaním dát zo serveru a práci s uloženými dátami získanými zo serveru, ktoré si sprístupní a načíta každá obrazovka podľa potreby.

3.4 Prístup k API zariadenia

Každá aplikácia, ktorá sa snaží využiť potenciál mobilných zariadení, vyžaduje prístup k senzorom zariadenia a ďalším informáciam. Ako dostať tieto údaje do javascriptového kódu a ako s nimi pracovať? Ako zabezpečiť funkčnosť na všetkých podporovaných platformách?

Riešenie: *PhoneGap* framework (kapitola 2.1) obsahuje sadu *Phonegap* a *Cordova pluginov*[7], ktoré slúžia na sprístupnenie *API*[8] (*web-to-native* abstraktnej vrstvy) zariadenia prostredníctvom JavaScriptu. Konkrétne sa jedná o *pluginy* na získavanie informácií o stave batérie, pre sprístupnenie kamery, kontaktov, informácií o zariadení, dáta z pohybového senzoru a smere zariadenie (kompas). Umožňuje prácu so súborovým systémom, vytvárať systémové notifikácie. Pre aplikácie pracujúce s geografickou polohou Cordova obsahuje plugin *Geolocation*. Ak je potrebné z aplikácie potrebné otvoriť obsah v inej inštancií webového prehliadača, slúži k tomu plugin *InAppBrowser*. Ďalšie pluginy sú pre prácu s médiami, sieťou či vibráciami. Vďaka API sa webová aplikácia mení na hybridnú[9] a často sa tento názov používa i v praxi. V mapových aplikáciach je väčšinou potrebné pracovať s geografickou lokáciou a v multiplatformových aplikáciach s pluginom pre získavanie informácií o zariadeniach.

V demonštračnej aplikácií je potrebné pracovať s geolokáciou a pluginom inAppBrowser, pre otvorenie Katalógu alebo nepodporovanej mapy v internetovom prehliadači. Plugin Device je potrebný pre získavanie informácií o zariadení a platforme.

[7] https://build.phonegap.com/plugins

[8] Pod skratkou *API*(*Application Programming Interface*) sa myslí rozhranie zariadenia pre programovanie aplikácií.

[9] http://blogs.telerik.com/appbuilder/posts/12-06-14/what-is-a-hybrid-mobile-app-

Kapitola 4

Výkon mobilnej aplikácie

Základnou požiadavkou na úspešnosť aplikácie je výkon. Pri tvorbe aplikácie je jej rýchlosť prvým a najdôležitejším kritériom. Ak sa jedná o mapovú aplikáciu, s ktorou používateľ nepretržite manipuluje, je to ešte dôležitejšie. Rýchlosť a plynulosť aplikácie veľmi súvisí s komfortom používateľa pri používaní aplikácie. Robert Miller objavil tri prahové úrovne ľudskej pozornosti [11]. Jedna desatina sekundy je vnímaná ako okamžite. Jedna sekunda je nutná aby používatelia cítili voľnosť pohybu v informačnom priestore. Doba odozvy by nemala prekročiť 10 sekúnd, lebo potom používatelia strácajú pozornosť. Táto kapitola čerpá zo zdrojov, na ktoré sa odkazujem v texte podkapitol.

4.1 Hardvérová akcelerácia animácii

Aby sme zabezpečili spomínanú voľnosť pre používateľa a zároveň získali čas na prednačítavanie ďalších stránok vo webových aplikáciach je vhodné použiť animácie. Animácia získa niekoľko milisekúnd kým sa používateľ začne sústrediť na zobrazovaný obsah. Aby však aplikácia mala možnosť pracovať počas animácie, je nutné zabezpečiť aby animácia nespotrebovala väčšinu procesorového času. Preto navrhujem časť úloh odsunúť na grafickú kartu. Napriek tomu, že popisované metódy akcelerácie sú určené pre 3D transformácie, navrhujem ich použiť i na prechody obrazoviek v 2D.

V takmer každej modernej mobilnej aplikácií sa vyskytuje množstvo transformácií a animácií. Vo webových aplikáciach k tomuto účelu využívame CSS. Jednou z možných ciest ako urýchliť beh je použiť hardvérovú akceleráciu CSS3. Táto sekcia čerpá praktické informácie z článku na Adobe Web Platform Team Blogu [12], článku z HTML5 Rocks[1][9] a z vlastných experimentov (6.3).

Pred samotným použitím akcelerácie je potrebné mať informácie ako vlastne akcelerácia pomocou grafickej karty funguje. Nemusí sa vždy jednať o výhody. Využitie grafického čipu (*GPU*) má vplyv na batériu a video-pamäť je tiež obmedzená.

Moderný webový prehliadač funguje obvykle v 2 vláknach. Obe vlákna spolupracujú pri vykresľovaní webovej stránky. Hlavné vlákno typicky zodpovedá za beh javascriptu, kalkuláciu CSS štýlov HTML prvkov, rozvrhnutie obsahu stránky, vykresľovanie elementov do bitovej mapy/máp a predávanie týchto máp druhému vláknu. Druhé vlákno - skladateľ (*compositor*) - je zodpovedné za vykresľovanie bitových máp na obrazovku pomocou GPU. Ďalej zisťuje, ktoré časti sú viditeľné, ktoré budú viditeľné (po rolovaní). Tiež žiada hlavné vlákno o bitové mapy týchto častí a zabezpečuje posúvanie stránky pri rolovaní. Hlavnému

[1] HTML5 Rocks je projekt vývojárov spoločnosti Google

vláknu môže trvať dlho reagovanie na používateľský vstup napríklad kvôli spracovávaniu javascriptu. Naopak *compositor* sa snaží reagovať okamžite. Snaží sa stránku prekresliť i 60-krát za sekundu ak sa mení. Napríklad pri rolovaní nečaká na bitové mapy doručené od hlavného vlákna, ale vykreslí čo môže a ostatné vyplní bielou farbou.

Compositor využíva grafický čip. Grafické čipy sú špecializované na konkrétne úlohy. V niečom sú skutočne rýchle:

- Vykreslovanie na obrazovku
- Vykreslovanie rovnakej bitovej mapy opakovane
- Vykreslovanie rovnakej bitovej mapy na inej pozícii, s rotáciou alebo zmenou veľkosti.

Naopak pomerne pomalou operáciou je nahrávanie bitovej mapy do pamäte, nakoľko prenos medzi procesorom a GPU je v mobilných zariadeniach viac limitovaný.

Ako to teda funguje s CSS? Príklad jednoduchej zmeny výšky elementu CSS s animáciou:

```
1 div {
2     height: 100px;
3     transition: height 1s linear;
4 }
5
6 div:hover {
7     height: 200px;
8 }
```

Na obrazku 4.1 je príslušný diagram ako to funguje. Červené bloky sú časovo náročnejšie. Zelené naopak rýchle. Na obrázku je vidieť, čo je najkritickejším krokom pri vykreslovaní.

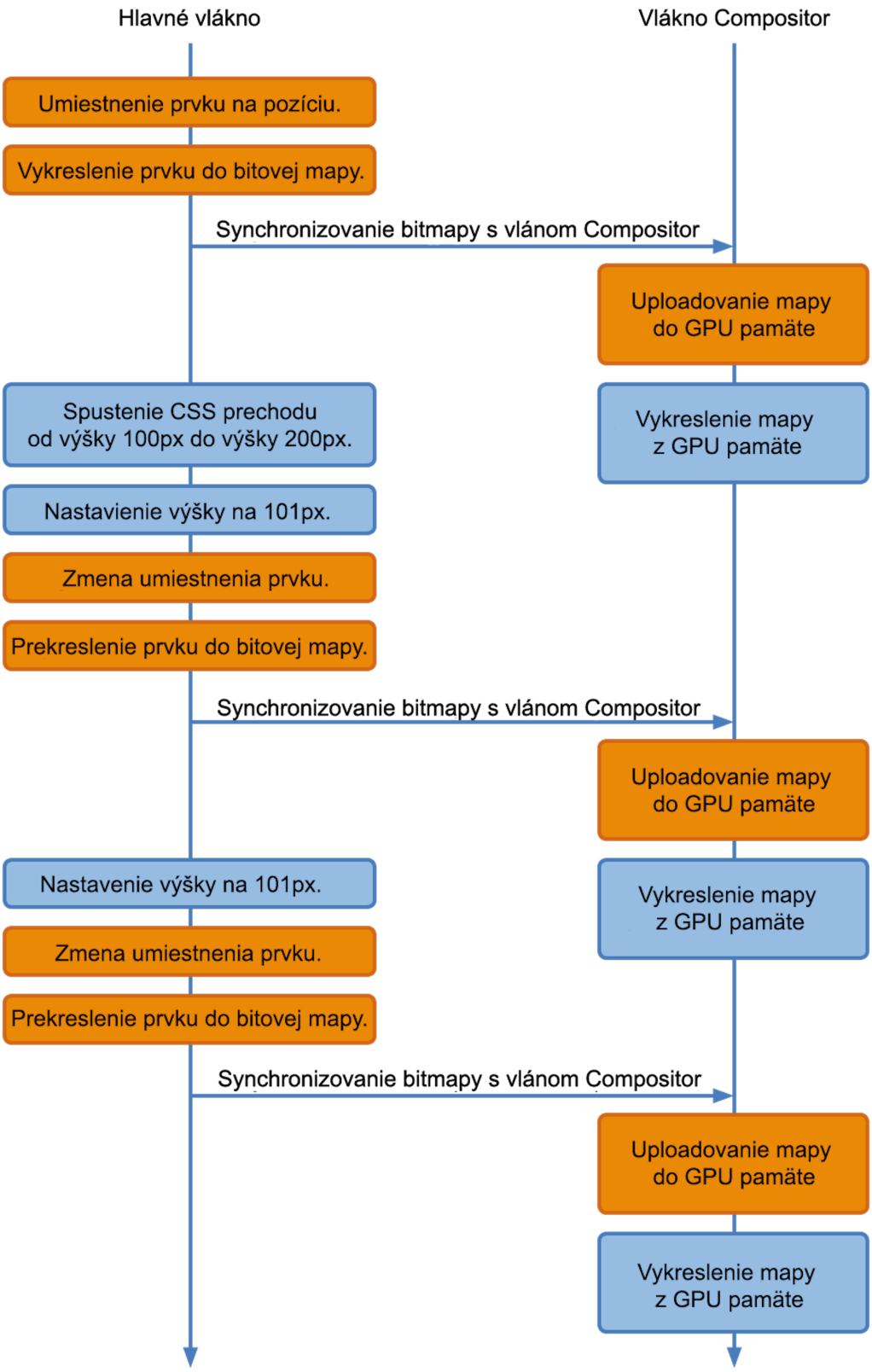
CSS nám však poskytuje modernejší prístup, a rovnaký efekt s animáciou dokážeme dosiahnúť s transformáciami pomocou kaskádových štýlov (*CSS*). Efektívnejší kód pre rovnakú zmenu a animáciu:

```
1 div {
2     transform: scale(0.5);
3     transition: transform 1s linear;
4 }
5
6 div:hover {
7     transform: scale(1.0);
8 }
```

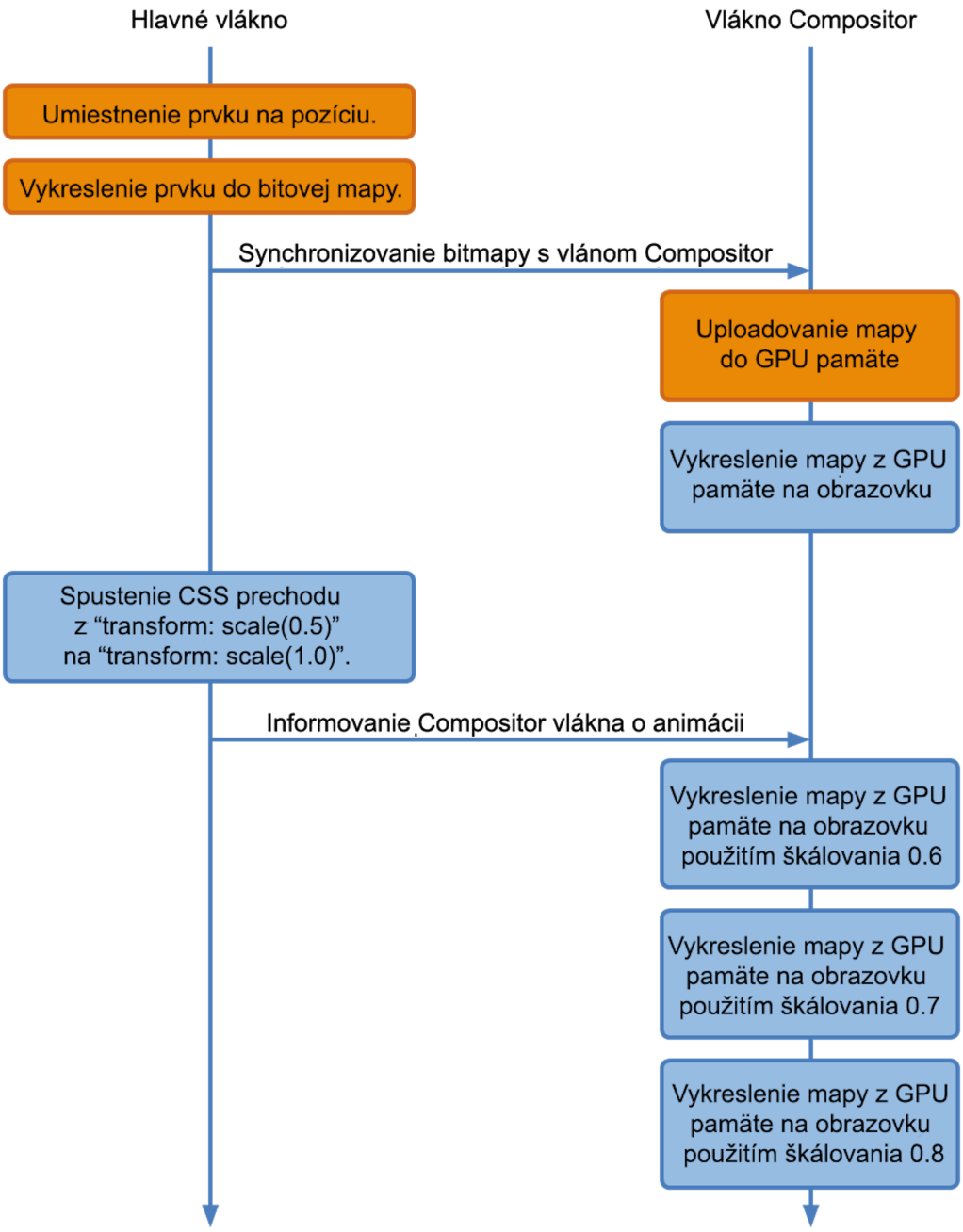
Na obrázku 4.2 je vidieť, že časovo kritických operácii podstatne ubudlo a teda zredukujeme i potenciálne trhanie animácie.

3D animácia v druhom prípade teda funguje tak, že na začiatku sa nahrá bitová mapa elementu do pamäte grafického čipu a následne sa už s ňou iba pracuje. Toto urýchlenie má zmysel u veľkých elementov, zložitých na prekreslenie. Týmto spôsobom ide akcelerovať tieto CSS vlastnosti:

- transform - posuny v osiach x,y,z v nadradenom prvku.
- opacity - priehladnosť 0 - neviditeľné, 1 - viditeľné, 0.5 - 50% priehľadnosť, ..
- fitre (blur, contrast, invert, ...) - záleží na konkrétnom filtre a webovom prehliadači



Obrázek 4.1: Diagram práce vlákien pri bežnom CSS[12].



Obrázek 4.2: Diagram práce vlákien pri CSS s použitím hardvérových akcelerácií[12].

Jednotlivé bitové mapy môžeme nazývať i vrstvy. Všetky animované elementy sú ukladané do osobitných vrstiev aby mohli byť animované nezávisle. V závislosti na prehliadači sú elementy rôzne zoskupované aby sa znížil počet vrstiev. Vo web-kit prehliadačoch sa vrstva vytvára pre každý animovaný objekt. Trikom ako zabezpečiť vlastnú vrstvu elementu je využiť CSS transformáciu:

```
translateZ(0)
// alebo
translate3d(0,0,0)
```

Tým pádom je možné zabezpečiť i hardvérovú akceleráciu na objekte, ktorý nie je hardvérom urýchľovaný automaticky.

Príliš mnoho vrstiev však môže spôsobiť problémy vo výkone či dokonca pády aplikácie.

Navrhované riešenie: Pre prípad mobilnej aplikácie navrhujem využiť hardvérové akcelerácie pre všetky animácie - prechody medzi obrazovkami, zobrazovanie menu, postranných panelov aj vyskakovacích okien. Aby nedošlo k vytvoreniu mnoho vrstiev, tak je vhodné zapnúť akcelerácie iba na animáciu. Navrhujem vytvoriť dve CSS triedy pre zobrazenú a dve pre skrytú obrazovku. Jedna trieda bude označovať pozíciu obrazovky a druhá bude mať pozíciu obrazovky nastavenú pomocou *translate3d(x,0,0)*, kde namiesto x bude pozícia obrazovky v x-ovej súradnici - napr. 100%. Pri udalosti, ktorá má skryť obrazovku, sa menia CSS triedy. Nastavenú máme triedu bez *translate3d()*, ktorá zabezpečuje zobrazenie obrazovky cez cele okno. Následne nastavíme hardvérom akcelerovanú triedu ktorá obsahuje *translate3d(0,0,0)* a *transition* pre zabezpečenie animácie. Tým sa manipulácia s daným oknom presunie do druhého vlákna. Následne nastavíme triedu ktorá obsahuje *translate3d(x,0,0);*, ta nám zabezpečí posun obrazovky. Pri zachytení udalosti konca animácie (*transitionend*) odstránime triedy hardvérovej akcelerácie a nahradíme ju triedou umiestňujúcou obrazovku na aktuálnej pozícii. Celý proces je možné obmeniť tak, že budeme používať iba 3 triedy, a v niektorej pozícii zanecháme obrazovku. Napríklad aktívna obrazovka bude vždy mať nastavenú triedu s vlastnosťou *translate3d()* ale je možne vymyslieť rôzne kombinácie v závislosti na počte obrazoviek.

Všetky animácie v aplikácií demonštrujú hardvérové aktualizácie. V kapitole 6.3 sú grafy ukazujúce namerané *FPS* akcelerovaných a neakcelerovaných animácií (počet snímkov za sekundu) 6.5-6.8.

4.2 Využitie potenciálu viacjadrových procesorov

Javascript ma jedno vláknové prostredie. Bežne nie je možné aby bežalo viac javascriptov súčasne. Pri vykonávaní časovo náročných operácií alebo získavaní dát zo serveru musí aplikácia - jej javascript čakať na odpoveď. V kapitole 4.1 je uvedené, že javascript beží v hlavnom vlákne. Teda môže sa ľahko stať, že pri zložitých operáciach stránka prestane reagovať na pokyny používateľa. Príkladmi takýchto operácií môže byť prednáčítavanie alebo ukladanie do vyrovnávacej pamäte (*cachovanie*) dát, práca s textom v reálnom čase, analýza obrazových alebo zvukových dát, dotazy na webové služby na pozadí, vstupno-výstupné operácie na pozadí, práca s veľkými dátovými štruktúrami či časovo náročné úpravy lokálnej databázy.

Aby sme dokázali zabezpečiť plynulosť používateľského rozhrania je nutné prevádzať dotazy v osobitnom vlákne. Technológia, ktorá umožňuje spustiť javascriptový súbor v osobitnom vlákne sa nazýva Web Worker[2]. Má to však určité obmedzenia.

[2]Špecifikácia http://www.w3.org/TR/workers/

Kapitola je založená na [6], Google projektu pre vývojárov [2] a informáciach získaných zo špecifikácií W3C konzorcia[3].

Pod pojmom Web Worker (alebo worker) bude popísaný dedikovaný web worker. Existuje i varianta zdieľaných web workerov, ktorá umožňuje komunikáciu workerov medzi sebou v rámci jednej domény, ale nie je podporovaná na Android zariadeniach. Web worker je javascript bežiaci v pozadí, nezávisle na ostatných skriptoch. Vzhľadom k tomu, že web worker beží v izolovanom vlákne, jeho javascript musí byť v individuálnom súbore. web workery by sa mali používať na úlohy s dlhou životnosťou, lebo ich spustenie je výkonnostne drahé a jedna inštancia pamäťovo náročná. workery komunikujú s ostatnými kontextami prehliadača prostredníctvom kanálov správ a ich portami (MessagePort definovaný v HTML5). Podľa W3C špecifikácie sa pri vytvorení workeru, pre skript s konkrétnou absolútnou URL adresou, vytvorí separátne vlákno, proces alebo ekvivalentná štruktúra. V súčasných implementáciách sa jedná vždy o vlákno. Od tejto chvíle bežia všetky ďalšie kroky asynchrónne v novovytvorenom kontexte. Načíta sa zdroj zo zadanej URL s nastaveným skriptovacím jazykom (javascript). Globálny objekt skriptu je globálny priestor workeru (worker global scope). Kontext prehliadača a objekt dokumentu sú prevzaté od vlastníka workeru.

Následujúci text ukazuje a popisuje použitie a možnosti web workerov: Prvým krokom je vytvoriť web worker pomocou konštruktoru:

```
1 var worker = new Worker('jsfile.js');
```

Ak zdrojový súbor existuje, webový prehliadač vytvorí nové vlákno, v ktorom začne načítavať javascriptový súbor. Kým nieje kompletne načítaný nezačne sa vykonávať. Po vytvorení web workera je potrebné ho spustiť pomocou metódy *postMessage*.

```
1 worker.postMessage(); // Start the worker.
```

Názov tejto metódy naznačuje ako prebieha komunikácia medzi hlavným vláknom a workerom - zasielaním správ. V moderných webových prehliadačoch je možné okrem textových reťazcov zasielať správami i JSON objekty.
Jednoduchý príklad Workera vyzerá nasledovne:
Main script:

```
1 var worker = new Worker('doWork.js');
2 worker.addEventListener('message', function(e) {
3   console.log('Worker said: ', e.data);
4 }, false);
5 worker.postMessage('Hello World'); // Send data to our worker.
```

doWork.js (the worker):

```
1 self.addEventListener('message', function(e) {
2   self.postMessage(e.data);
3 }, false);
```

Obe skripty obsahujú *EventListener*, ktorý čaká na udalosť *message*, na základe ktorej vykonajú príslušný kód - v hlavnom vlákne sa jedná o výpis do konzoly a vo workeri na prijatú správu odpovedá opäť správou pomocou metódy *postMessage()*. Avšak data pri posielaní správ niesú zdielané ale su kopírovane - sú serializované odoslané a deserializované. Väčšina moderných webových prehliadačov prevádza kódovanie a dekódovanie JSON objektov automaticky. Worker sa dá ukončiť dvoma spôsobmi - z hlavného vlákna:

```
1 worker.terminate()
```

alebo vo vnútri workera ako:

[3] http://www.w3.org/TR/workers/

```
self.close()
```

Občas je potrebné preniesť medzi workerom a hlavným vláknom iný typ dát. Väčšina prehliadačov má implementované algoritmy pre štruktúrne klonovanie - to umožní preniesť dáta ako File, ArrayBuffer, Blob ci JSON objekt. Avšak často sa môže jednať o väčší objem dát a kopírovanie môže zabrať pár milisekúnd. Na takéto prípady je možné použiť *Transferable Objects*.

Transferable Objects sú prenášané z jedného kontextu do druhého, dá sa to predstaviť ako predanie referencie na objekt v jazyku C/C++, s tým rozdielom, že v starom kontexte workeru po prenose už dostupné nie sú. Na použitie *Transferable Objects* sa používa tiež metóda *postMessage()* ale s odlišnými parametrami:

```
worker.postMessage(arrayBuffer, [arrayBuffer]);
window.postMessage(arrayBuffer, targetOrigin, [arrayBuffer]);
```

Prvý parameter sú dáta, ktoré chceme preniesť a druhý je zoznam prvkov, ktoré by mali byť prenesené.
Web Workery majú prístup iba k obmedzeným možnostiam javascriptu:

- Objekt Navigator
- Object location (čítanie)
- XMLHttpRequest
- setTimeout()/clearTimeout()
- setInterval()/clearInterval()
- Cache aplikácie
- importScripts() metóda pre načítanie externých súborov
- vytváranie ďalších workerov

a nemajú prístup k:

- Stromu DOM
- Objekt „window“
- Objekt „document“
- Objekt „parent“

Použitie web workerov je vhodné v prípadoch, ktoré som už spomínal v definícii problému:

- Predbežné načítavanie alebo cachovanie dát
- Formátovanie či kontrolovanie textu v reálnom čase
- Analýza obrazových alebo zvukových dát
- Dotazy na webové služby na pozadí
- Vstupno-výstupné operácie na pozadí

- Predspracovanie alebo spracovanie veľkých dátových štruktúr (napr. JSON objektov)
- Časovo náročné úpravy lokálnej databázy

Navrhované riešenie: Vo veľkom počte webových aplikácií, ako napríklad v mapových aplikáciach, je potrebné komunikovať so serverom. V tomto prípade navrhujem použiť web workery práve pre túto pasáž, kedy nie je možné určiť akú dobu bude táto komunikácia trvať a často získané dáta vyžadujú dodatočné spracovanie. Samozrejme, že pri využití technológií ako AJAX alebo JSONP dochádza k asynchrónnej komunikácií, ale neblokujúce neznamená konkurentné. Navyše získané dáta často sú ďalej spracovávané ako napríklad vkladané do šablón, do výpočtov alebo môžu obsahovať url obrázkov, ktoré je potrebné predbežne načítať. Takže využitie web workerov má zmysel i v tomto prípade. Použiť túto technológiu je vhodné hlavne v zložitých dotazoch, ktoré môžu spôsobiť problémy pri interakcii s používateľským rozhraním. V prípade AJAX i JSONP technológií sa však stretneme s problémami. V prípade AJAX technológie problém nie je s technológiou web workerov, ale s komunikáciou so vzdialeným serverom. Je nutne povoliť CORS (cross-rigin requests). Viac o tomto probléme sa nachádza v kapitole 3.1. Technológia JSONP[4] vznikla práve preto, aby umožnila komunikáciu medzi rôznymi servermi teda i lokálnym (mobilné zariadenie) a vzdialeným. Technológia JSONP pracuje tak, že vytvára HTML *script* prvok na strane zasielajúcej dotazy a teda požaduje prístup k stromu DOM. V predchádzajúcom texte je uvedené, že web workery túto možnosť nemajú. Napriek tomu web workery umožňujú importovanie externých javascriptových súborov pomocou metódy *importScripts()*. Navrhujem rovnakú metódu použiť i na externý skript. Ukážka ako vytvoriť triedu založenú na *goog.net.Jsonp*, aby bola použiteľná vo web workeroch sa nachádza v prílohe D. Najväčším nedostatkom je, že metóda *importScripts()* neumožňuje platformovo univerzálnym odchytiť vzniknuté chyby komunikácie ako napríklad prerušenie spojenia. Z tohto dôvodu je vhodnejšie využiť AJAX s CORS hlavičkami, ale v prípade, že z nejakého dôvodu (prístup, nutná podpora starších technológií, bezpečnosť) nie je možné prevádzať úpravy na vzdialenom serveri, tak je možné použiť i JSONP. Riešenie spomínaných problémov je už ale závislé na konkrétnom prípade použitia.

Dotazovanie a práca so serverovými dátami demonštračnej aplikácie využíva web workery spolu s predbežným načítavaním obrázkov. Ak by web workery použité neboli, tak by napríklad pri práci s používateľským rozhraním mohol byť prijatý výsledok a začal sa spracovávať - načítavať obrázky zo serveru. Mohlo by dôjsť k zhoršeniu interakcie s používateľským rozhraním. Viac v kapitole 6.3(str. 53), v časti o testovaní viacerých vláken.

4.3 Moduly v JS

Ďalšia z možných otázok pri tvorbe javascriptu je ako štrukturizovať súbory javascriptu? Ako zamedziť načítavaniu množstva javascriptového kódu pri štarte webových aplikácii? Ako zabezpečiť aby sa časť javascriptu načítala až vo chvíli, keď je vyžadovaná?

Riešením je modulácia. Ako dopomôže modulácia webovej aplikácii? Ak sa spustí aplikácia, je možné načítať iba niektoré moduly - iba javascript, ktorý je momentálne potrebný. Urýchli to spustenie aplikácie i načítavanie jednotlivých obrazoviek. Moduly majú ešte väčší význam v použití v rámci iframov, kedy sa nemusí všade a vždy načítavať celý javascript. Každý iframe ma vlastný modul - klasický javascriptový súbor. Vo chvíli, keď bude vytvorený iframe, načíta sa i tento javascript. Od modulov sa ale očakáva ešte viac. Ako

[4] http://www.json-p.org

zabezpečiť aby sme mohli javascript načítať v čase keď je naozaj potrebný? Táto kapitola je založená na [4].

Prvý druh modulácie vychádza z použitia iframov (kapitola 3.3), kde vlastne každý iframe, môže obsahovať vlastný unikátny javascriptový súbor, ktorý sa načíta iba s inicializáciou - vytvorením iframe-u. Ale aj tak vo veľkom množstve prípadov nastane, že v jednotlivých iframeoch sa nachádza rozsiahly kód, ktorý by mohol byť osobitným modulom dodatočne načítavaným podľa potreby. K tomu slúžia moduly podporované knižnicou Google Closure Tools.

Plovr[5], prekladací nástroj knižnice *Google Closure*, umožňuje moduláciu javascriptu. Automatizovaná tvorba modulov pri preklade na základe závislosti a správa modulov podporovaná priamo triedami Closure Tools dáva modulácií javascriptu čistejšiu a jednoduchšiu formu. Napríklad môžme načítať modul pri kliku na tlačítko alebo pri zavolaní určitej funkcie. V zložitejšej aplikácií s rozsiahlejšími skriptami sa musí uplatniť nejaká predikcia a predikcia načítavania. Je to plne v správe programátora. Ďalšou výhodou pri rozsiahlom module je, že sa načítava iba raz a je možné ho použiť i v ostatných moduloch, súboroch. Pri preklade javascriptu pomocou prekladača vznikne pre modul osobitný javascriptový súbor a k nemu inicializačný súbor, ktorý riadi načítavanie modulu v správnej chvíli.

Pri použití Plovru, sa dajú moduly vytvoriť zadefinovaním pri preklade, nastavením závislostí, a následne budú automaticky načítané podľa potreby. Kód je založený na dokumentácií Plovru[6].
Príklad parametrov prekladu:

```
"modules": {
    "modul1": {
      "inputs": "src/modul1_init.js",
      "deps": []
    },
    "modul2": {
      "inputs": "src/modul2_init",
      "deps": "modul1"
    }
  },
  "module-output-path": "deploy/module_%s.js"
```

Pre podporu modulov Google Closure poskytuje tieto dve triedy:

- *goog.module.ModuleLoader* - načítava javascriptové moduly
- *goog.module.ModuleManage* - udržuje informácie o všetkých moduloch v prostredí. Vzhľadom k tomu, že moduly nemusia mať načítaný kód, musia byť sledované.

Inicializácia modulu sa zapisuje do pomocného suboru napr. *modul_init.js*, ktorý slúži na inicializácie modulov - napríklad volanie konštruktoru. Pomocou metódy *setLoaded()*, môže po inicializácií a načítaní javascriptu modul manažérovi oznámiť, že bol načítaný. V súbore *modul.js* je potom implementácia modulu. Pre lepšie pochopenie je vhodné nahliadnuť do mojich demonštračných príkladov[7], ktoré sú zverejnené v rámci ukážkových aplikácií pre použitie *Google Closure* na *GitHub* účte spoločnosti Klokan Technologies.

[5] http://plovr.com/
[6] http://plovr.com/modules.html
[7] Príklady použitia modulov:
https://github.com/klokan/closure-library-plovr-hello-world-skeletons/tree/master/modules,
https://github.com/klokan/closure-library-plovr-hello-world-skeletons/tree/master/modules-api

V demonštračnej aplikácií má každá obrazovka/iFrame rámec vlastný javascriptový modul. V rámci týchto skriptov však stojí za zváženie modulácia rozsiahlych skriptov - napr. mapové prehliadače, ktoré by nemuseli byť načítané hneď pri vytvorení objektu. Rovnako sa vo všetkých javascriptoch (každej z obrazoviek) vyskytuje objekt pre sledovanie veľkosti obrazovky či rotácie a objekt pre komunikáciu medzi iFrame rámcami. Napriek tomu, že mapové prehliadače majú rozsiahly javascript, navrhujem skôr obetovať čas pri načítaní aplikácie za cenu bezproblémového behu aplikácie. Každopádne mapová prehliadač sa nachádza hneď na úvodnej obrazovke, takže v tejto aplikácii by to aj tak nemalo veľký význam.

4.4 Šablóny

V mobilných aplikáciach sa často stretávame s rovnakým rozložením elementov, ale odlišným obsahom. Napríklad pre zobrazenie výsledkov v rolovacom zozname (zoznam výsledkov vo vyhľadávači Google, zoznam príspevkov na Facebooku a pod.). Inokedy je potrebné zobraziť rovnaký obsah na rôznych miestach - napríklad menu na rôznych obrazovkách. Možné je v tomto prípade javascriptom zapisovať HTML elementy na konkrétne miesta, ale aj v tomto prípade je pre pohodlnejší vývoj a jednoduchšiu údržbu lepšie použiť šablóny. Táto podkapitola sa nachádza v kapitole o výkone z iného dôvodu ako predchádzajúce. Pri nesprávnom použití totiž dochádza k zníženiu výkonu aplikácie. Ako navrhujem správne používať šablóny je popísané v nasledujúcom texte. Časť venovaná Google Closure Templates vychádza z [4].

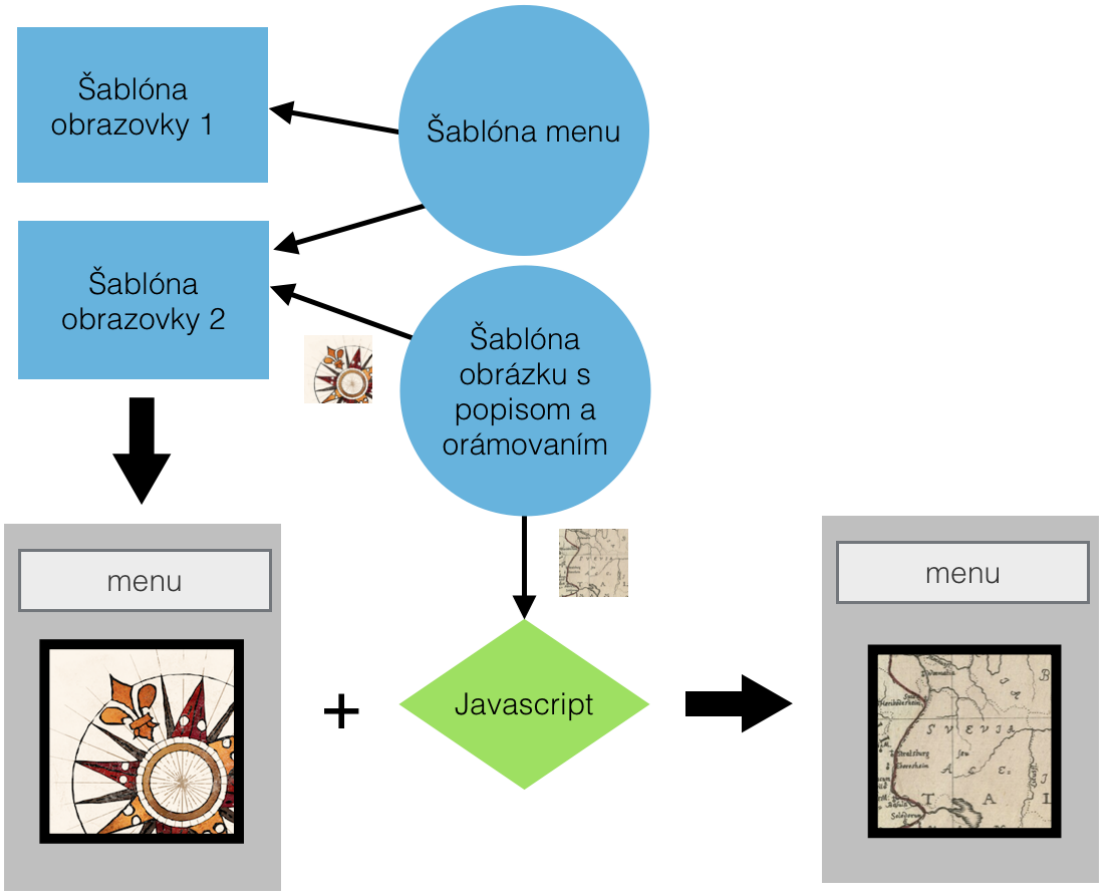
V takmer každej aplikácií by malo platiť, že štýl a rozloženie obrazoviek je zhodné. Pre vývoj, úpravu a údržbu je nepraktické mať pre každú stránku podobný kód a preto je vhodnejšie vytvoriť šablónu s parametrizovateľným obsahom, ktorá bude spoločná pre viacero obrazoviek. Problémom zostáva nástroj. V kapitole 2.3 sa píše o *Google Closure Tools* nástrojoch. Medzi nich patrí i *Google Closure Templates*. Prostredníctvom *SOY* súborov je možné slobodne tvoriť HTML, CSS i iné šablóny. Ukážka takejto šablóny sa nachádza na obrázku 4.4.

Podpora šablón a ich použitie nemusí vždy viesť k zjednodušeniu a zrýchleniu aplikácie. Práve naopak, použitie šablón často zvádza k aktualizácií obsahu opätovným použitím šablóny s novými parametrami. Oproti zmene jednotlivých HTML prvkov to vedie k spomaleniu aplikácie. K efektívnemu použitiu je potrebný podrobný návrh pre konkretnú aplikáciu. V tomto texte popíšem ako navrhujem používať šablóny v najčastejších prípadoch.

Vždy pri tvorbe aplikácie je potrebné vytvoriť hierarchiu obrazoviek. Šablóna z Google Closure Templates je po kompilácii javascriptová funkcia s parametrami a návratovou hodnotou v podobe reťazca (HTML kódu). Pri tvorbe obrazovky sa použije šablóna s počiatočnými parametrami. Z toho vyplýva, že bude existovať jedna šablóna pre každú obrazovku s voliteľnými parametrami, upravujúcimi stav obrazovky. Voliteľný parameter počiatočného stavu môže byť napríklad meno prihláseného používateľa, aktuálna poloha alebo dátum. Na viacerých obrazovkách jednej aplikácie sa často nachádzajú spoločné prvky ako napríklad menu alebo stavová lišta. Hierarchia šablón nám umožňuje šablóny do seba zanorovať a používať na viacerých miestach. Môžme mať pre každú obrazovku rôznu šablónu. ale v rámci každej budú použité šablóny spoločných prvkov.

Pre výkon aplikácie je najdôležitejšie neprekresľovať celú obrazovku pri zmene časti obsahu. Prekreslením celej obrazovky by sa aplikácia spomalila a navyše by vznikol nepríjemný

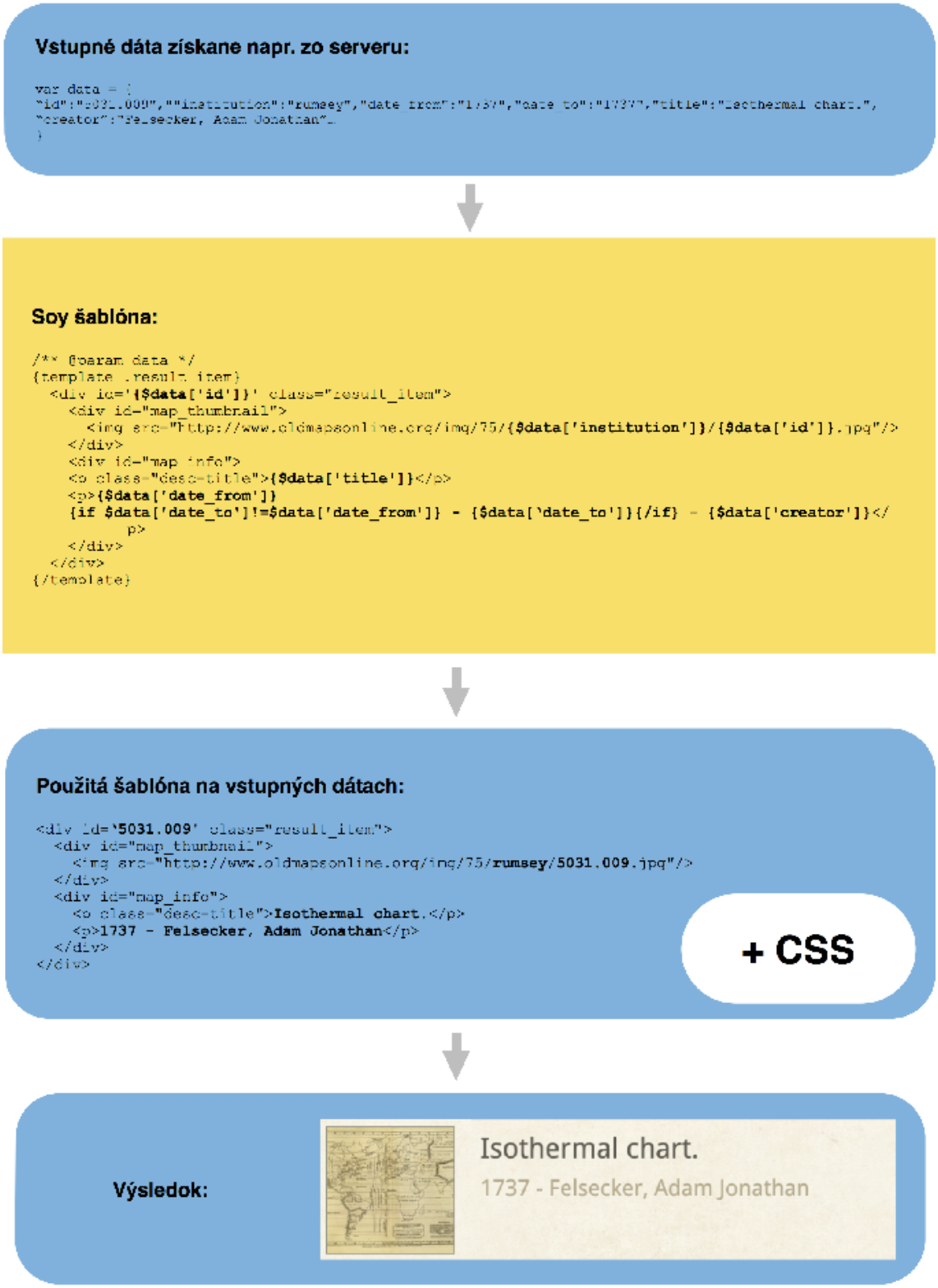
efekt bliknutia. Ak je potrebné zmeniť zložitejší[8] prvok stránky, je vhodné mu vytvoriť novú šablónu. Táto šablóna môže byť použitá v rámci inej šablóny, a zároveň ju je možné použiť i v javascripte pre vlastnú potrebu. Pri použití v javascripte je vygenerovaný HTML kód vo forme reťazca, ktorým môžme nahradiť predchádzajúci prvok stránky alebo ho pridať ako nový. Týmto prístupom dosiahneme elegantne udržiavateľný kód s jednoduchým použitím bez straty výkonu. Popis uvedeného príkladu lepšie znázorňuje obrázok 4.3.



Obrázek 4.3: Ukážka navrhovaného použitia šablón a aktualizácie obsahu obrazovky

V demonštračnej aplikácií pre všetky šablóny sú použité Google Closure templates (kapitola 2.3). Každá obrazovka má vlastnú šablónu. Pre jednotlivé opakujúce elementy (lišty, menu a pod.) sa využívajú ďalšie šablóny, ktoré sú zdieľané v rámci šablón jednotlivých obrazoviek. Pre aktualizované výsledky rovnako demonštračná aplikácia využíva šablóny výsledkov, ktoré sú iba naplnené relevantnými dátami.

[8] Pod pojmom zložitejší sa myslí prvok, ktorý má zložitejšiu štruktúru - niekoľko HTML prvkov. Napríklad text na stránke môžeme zmeniť priamo bez potreby šablóny.



Obrázek 4.4: Ukážka použitia šablóny v demonštračnej aplikácií

Kapitola 5

Používateľské rozhranie a interakcia

Predmetom tejto kapitoly je ovládanie a prispôsobenie používateľského rozhrania potrebám používateľa, možnostiam zariadenia a technológiám webových aplikácií. Cieľom je čo najviac používateľské rozhranie pripodobniť natívnemu.

V prvej podkapitole sa rieši interakcia s mapou a podobnými prvkami. Jedná sa o mohutné ovládacie prvky s niekoľkými možnosťami interakcie. Text sa zaoberá možnosťou pridania ďalších reakcií na konkrétne udalosti. Ďalšia podkapitola sa zaoberá detekciou zariadení z pohľadu používateľského rozhrania - ako prispôsobovať dizajn rôznym zariadeniam. Ďalšia podkapitola sa týka vizuálnych rozdielov medzi webovými a natívnymi aplikáciami a ako ich zmenšiť. Posledná sekcia sa hlbšie zameriava na dotykové gestá a udalosti.

5.1 Interakcia s mapou a inými zložitejšími prvkami

V uživateľských rozhraniach, v ktorých pracujeme s mapou, je niekedy potrebné vykonávať nejakú činnosť. Príkladom môže byť zobrazovanie elementov vo výreze alebo filtrovanie dát na základe výrezu. Ak sa táto činnosť vykonáva počas pretrvávajúcej interakcie s mapou, môže dochádzať k blokujúcemu volaniu rôznych javascriptov. Následkom je zníženie presnosti a plynulosti ovládacieho prvku. Problematika popísaná v tejto kapitole sa netýka len interakcie s mapou. Podobné správanie môžu mať všetky prvky, ktoré ovládame gestami, ktoré trvajú určitý čas - ťahanie, posúvanie alebo zväčšovanie. Príkladom môžu byť rôzne posuvníky. Rovnakým prípadom môže byť i činnosť vykonávana počas zadávania textu. Príkladom môže byť vyhľadávanie v Googli, ktorý vyhľadáva výsledky už počas písania dotazu.

Všeobecné riešenie: Riešením počas interakcie s mapou je vykonávať len tak zložitý kód, aby uživateľské rozhranie bolo stále dostatočne reaktívne. Druhou možnosťou je odsunúť vykonávaný kód do iného vlákna pomocou Web Workerov 4.2 - bez prístupu k DOM a ďalším objektom. Ďalšou špecifickou variantou môžu byť hardvérove CSS3 akcelerácie 4.1, ktorými je možné odsunúť transformácie HTML prvkov do druhého vlákna.

Čo ak nie je možné alebo vhodné použiť ani jednu z uvedených variant? Napríklad použitie mapového prehliadača, ktorý už využíva hardvérové akcelerácie, avšak pri interakcii vykonáva i inú činnosť. Dôsledkom interakcie sa pracuje s DOM - nie je možné použiť web workery. Tento problém navrhujem riešiť tak, že uprednostním plynulosť mapy (resp. iného prvku). Činnosť, ktorá je vykonávaná v dôsledku interakcie s mapou je potlačená

a zobrazenie výsledkov oneskorené. To navrhujem riešiť tak, že daná činnosť sa vykoná až po skončení interakcií s mapou. Možnosti sú dve. Jednou môže byť časové oneskorenie, ktoré je každou ďalšou akciou reštartované. Činnosť sa vykoná až po určitej dobe bez zmeny mapy. To ale stále môže spôsobiť i spomalenie interakcie pod pevný časový interval. Navyše ak používateľ znovu začne pracovať s mapou uprostred neprerušiteľnej činnosti (po zaslaní požiadavku na server), dôjde k spomínaným problémom s mapou. Druhou možnosťou je reagovať na udalosti používateľského rozhrania - dotyk, klepnutie, ťahanie, pustenie a pod. Pomocou týchto udalostí je možné zaznamenať začiatok i koniec interakcie. Koniec je potrebný pre spustenie činnosti. Začiatok je možné využiť na detekciu opätovnej interakcie a vyvolať prerušenie vykonávanej činnosti.V intervale medzi udalosťami by sa nemalo diať nič časovo náročné[1]. Ak to aplikácia umožňuje navrhujem použiť kombináciu udalosti i s časovačom.

Táto technika je v demonštračnej aplikácií použitá ako na interakcii s mapou, tak i s časovou osou a pri textovom filtrovaní pomocou atribútov.

5.2 Responzívny dizajn a detekcia zariadení

S variabilitou rôznych zariadení a rôznych rozlíšení obrazoviek je potrebné navrhnúť dizajn, ktorý bude použiteľný na každom zariadení. Niekedy je potrebné aby bol na dvoch zariadeniach podobný (rôzne mobilné telefóny), inokedy aby bol úplne rôzny (mobilné telefóny a tablety). Ako túto prispôsobiteľnosť zariadenia zabezpečiť?

Riešenie: Pri dizajne webovej aplikácie nemôžme použiť rovnaký HTML kód pre všetky mobilné zariadenia, ale CSS kód často musí byť pre rôzne platformy z technického hľadiska mierne upravený. Na rôznych platformách sú často veľké rozdiely pri rovnakom kóde. Priestorové rozdiely sa týkajú rôznych rozlíšení obrazoviek. Technické rozdiely sú spôsobené rôznou funkčnosťou alebo nefunkčnosťou jednotlivých CSS vlastností. Druhé hľadisko je používateľské - aplikácia na menších zariadeniach (mobilné telefóny) má často výrazne zjednodušené používateľské rozhranie ako pri väčších zariadeniach (tablety). Aby sme vyriešili problémy s rôznymi rozlíšeniami zariadení je nutné navrhnúť responzívny dizajn[2]. Responzívny dizajn má flexibilnú štruktúru, ktorá je väčšinou riešená prostredníctvom CSS a percentuálnymi rozmermi elementov. Podobne sa to rieši i s obrázkami, ktoré majú zadané percentuálne rozmery v rámci nadradených elementov. Ďalšou dôležitou časťou pri responzívnom dizajne sú Media Queries, pravidlá CSS3, ktoré umožňujú aplikovať odlišný dizajn na základe rozmerov zariadenia a jeho ďalších atribútov. Kapitola vychádza z [7].

Detekcia je potrebná i v rámci použitia správnej ikony a štartovacej obrazovky. Na to je možné použiť framework, ktorý zaobaľuje webovú aplikáciu do natívnej. Phonegap používaný v demonštračnej aplikácie nie je výnimkou. Ako nastaviť ikony a štartovacie obrazovky vo PhoneGap[3] je možné nájsť v prílohe C.

Pre optimalizáciu aplikácie pre rôzne zariadenia je potrebné občas detekovať zariadenie, o ktoré sa jedná i mimo CSS. Napríklad ak určitá funkcia má byť dostupná iba na tabletoch. Táto detekcia prebieha v rámci javascriptu, k tomu slúži plugin Device z PhoneGap[4]. Ďalšou možnosťou detekcie pomocou javascriptu je parsovanie reťazca získaného z objektu Navigator ako *navigator.userAgent*.Tiež je možné použiť metódy triedy *goog.userAgent* v rámci

[1] Rozhodnúť čo je časovo náročné je potrebné podľa konkrétnej aplikácie. Stolový počítač a low-end mobilný telefón majú túto hranicu v úplne odlišných číslach.

[2] Názov *responzívny dizajn* použil prvýkrát na blogu *A List Apart* Ethan Marcotte [10]

[3] Popisovaný Phonegap framework je vo verzii 3.3.0

[4] http://docs.phonegap.com/en/3.3.0/cordova_device_device.md.html#Device

Google Closure Tools.

V demonštračnej aplikácií sa táto detekcia týka používateľského pohľadu - rozdiely medzi tabletmi a menšími zariadeniami v rámci CSS, odlišné ikony aj štartovacie obrazovky pre rôzne zariadenia a platformy. Potrebná je i detekcia rôznych platforiem. Na iOS je potrebné počítať so stavovou lištou, ktorá napríklad u Androidu nie je súčasťou aplikácie.

5.3 Prednačítavanie obrázkov

Niekedy je vhodné použiť triky na prednačítavanie obrázkov. Tým sa zabezpečí, že v dobe zobrazenia, bude obrázok pripravený. Význam ešte viac rastie, ak sú obrázky na inom serveri a získavanie obrázkov môže trvať dlhšiu dobu. Jedna z možnosti je načítať obrázky neviditelné (Vlastnosť CSS *opacity:0;*. Použitie *display:none;* a *visibility:hidden;* v prehliadačoch často zabráni načítaniu obrázkov). Po načítaní je možné v potrebnej chvíli obrázky zobraziť.

Inou možnosťou je CSS3 načítavanie do vyrovnávacej pamäte (*cache*), kedy sa obrázky načítavajú mimo obrazovky a zároveň prednačítavajú do pamäte *cache* webového prehliadača. Ak v sa inom prvku použije rovnaká adresa k obrázku, tak sa použije ten prednačítaný.

Na moderných prehliadačoch ešte existuje možnosť použiť *XMLHttpRequest2* (*XHR2*), ktorý umožňuje i prácu so súbormi. Následujúci text je založený na [3]. V následujúcom kóde je krátka ukážka ako načítať obrázok do typu *Blob* (*Binary large object*)[5]. Blob je dátový typ používaný v databázach pre bližšie nešpecifikované binárne dáta.

```
1  var xhr = new XMLHttpRequest();
2  // !!! Vzdialeny server musi mat nastavene CORS hlavicky
3  xhr.open('GET', '/cesta/obrazok.png', true);
4  xhr.responseType = 'blob';
5
6  xhr.onload = function(e) {
7    if (this.status == 200) {
8      var blob = new Blob([this.response], {type: 'image/png'});
9      // ...
10   }
11 };
12
13 xhr.send();
```

S takto získaným objektom môžme po načítaní ďalej pracovať.

Techniku s *XHR2* navrhujem použiť ako trik pre prednačítavanie obrázkov v osobitnom vlákne - použitie vo *web workeri* (kapitola 4.2). Zo spomínaných možností, je použitie *XHR2* jediné riešenie, ktoré nepracuje s DOM stromom, preto ho je možné vo web workeri použiť. Ak sa obrázky získavajú zo vzdialeného serveru je nutné mať povolené *CORS* (*cross-origin-requests*). Viac o CORS v kapitole 3.1. Vo web workeri sa načítajú všetky obrázky pomocou kódu uvedeného vyššie. Web worker následne zašle správu, že načítanie obrázkov bolo dokončené. V tej chvíli môže hlavné vlákno začať pracovať s obrázkami, ktoré už budú dostupné v cache pamäti prehliadača.

5.4 Rozdiely medzi natívnymi a webovými aplikáciami

Táto kapitola sa konkrétne zaoberá vyrovnávaním rozdielov používateľského rozhrania natívnych a webových aplikácií. Obsahuje niekoľko odporúčaní, ktoré je vhodné dodržovať pre

[5] http://www.w3schools.com/sql/sql_datatypes.asp

optimálne zobrazenie a lepšiu používateľskú interakciu.

Optimalizovať vykresľovanie pri aktualizácii dát

Pri získavaní a aktualizovaní dát dochádza k vykresleniu/zobrazeniu HTML elementov. Tieto elementy sú vykresľované často po častiach a v náhodnom poradí. Načítanie polovice obrázku a tak donačítanie druhej časti, alebo načítanie rámu obrázku s následným donačítaním obrázku je veľmi rušivé. Inokedy zase dochádza k zmene pozície elementu počas načítavania.

K vizuálnym záležitostiam spojeným s vykresľovaním dát je opäť možné použiť CSS triky ako je popísané v kapitole 5.3 u prednačítavania obrázkov. Trik s priehľadnosťou je možné použiť i na iné prvky ako obrázky. Vlastnosť *opacity:0* na HTML prvku nezviditeľní daný objekt i všetkých potomkov v DOM strome. Problémom zostáva ako informovať element, že všetky elementy sú už úspešne načítané a pripravené k zobrazeniu. Objekty HTML ako napr. *object* či *img* obsahuje udalosť *onload*, ktorú je možné odchytiť na každom požadovanom objekte. Vo chvíli keď sú všetky pripravené, je možné zrušiť priehľadnosť. S pridaním animácie je výsledok vizuálne príjemnejší. Napríklad:

```
1     transition: opacity 200ms linear;
```

Používať 3D transformácie

Plynulosť animácie/transformácie dosiahneme hardvérovo akcelerovanými animáciami - kapitola 4.1.

Zotrvačné rolovanie

Riešenie zotrvačného rolovania je popísané v kapitole 5.5.

Dodržovanie maximálnej veľkosti textúr

Textúry sú často dátovo najnáročnejšou súčasťou používateľského rozhrania. Navrhujem dodržiavať niekoľko pravidiel priamo z doporučenia iOS Developer stránok[6] a aplikovať ich i na webové aplikácie:

- Aplikovať textúry pri spustení aplikácie, nikdy ich nemeniť počas behu aplikácie.
- Redukovať veľkosť pamäte, ktorú textúry zaberajú.
- Kombinovať menšie textúry na vytvorenie väčšej.

Skrývanie a vypínanie veľkých obrázkov a elementov s 3D transformáciou vždy, keď je to možné

Veľký počet veľkých obrázkov môže spôsobovať pády aplikácie. Preto je potrebné tie, ktoré sú mimo obrazovky alebo pod inými elementami skrývať pomocou *display: none;* alebo *visibility: hidden;*. Podobne je to s objektami s 3D hardvérom akcelerovanými transformáciami a vlastne nie je na škodu skrývať všetky elementy. Vypnutím týchto elementov dôjde k výraznému zrýchleniu a plynulosti aplikácie.

[6] https://developer.apple.com/library/ios/documentation/3ddrawing/conceptual/opengles_programmingguide/OpenGLES_ProgrammingGuide.pdf

Dotykové zvýrazňovanie

Pri dotykovom ovládaní webových stránok sa, vo web-kit prehliadačoch, daný element zvýrazňuje šedým podsvietením. Tento nenatívny efekt sa dá odstrániť pomocou nastavenia farby na priehľadnú, prípadne inú požadovanú farbu:

```
-webkit-tap-highlight-color:rgba(0,0,0,0);
-webkit-tap-highlight-color:transparent; // alternatívny zapis
outline: none;
```

Označovanie obsahu

Pri dlhom dotyku v stránke zobrazovanej vo web-kite sa označí príslušný obsah, ktorý je možné kopírovať a pod. Tento jav sa dá odstrániť nastavením tejto CSS vlastnosti:

```
-webkit-touch-callout: none; // zabrani zobrazeniu menu s~moznostou
    kopirovania
-webkit-user-select: none;
```

Výčet ďalších jednoduchých nastavení prostredníctvom Phonegap 3.3.0 frameworku s existujúcimi HTML alternatívami, ktoré sú potrebné pre dosiahnutie natívneho vzhľadu sa nachádza v prílohe C.

5.5 Ovládanie dotykových aplikácií

Dotykové ovládanie je jednou zo základných zmien s príchodom dotykových obrazoviek. Dotykové obrazovky sa najviac presadili v oblasti mobilných zariadení. Preto je potrebné webové aplikácie tomuto ovládaniu prispôsobiť. Táto kapitola sa zaoberá riešením dotykového rolovania dlhých zoznamov a podpore na rôznych zariadeniach. Ďalšie sekcie sú venované navigácií medzi jednotlivými obrazovkami, vrátane prepínania obrazoviek pomocou ťahania (*snap* gestá). Aplikácia čerpá z [5].

Rolovanie

V natívnych mobilných aplikáciach sú uživatelia zvyknutí na zotrvačné rolovanie, ktoré však vo webových aplikáciach natívne nieje, respektíve je iba na HTML prvku *body.*

U iOS a najnovších Androidov stačí nastaviť *-webkit-overflow-scrolling: touch* pre natívne rolovanie, ale nefunguje na iOS5. Pre iOS5 je dostačujúce ponechať pôvodné správanie, nakoľko k 4. máju 2014 len 2% používateľov používajú verziu iOS 5 a nižšiu[7]. U Androidu je to komplikovanejšie - neexistuje 100% funkčné riešenie, dokonca *overflow:auto* a *overflow:scroll* funguje ako *overflow:hidden* na Android 2.X. Verzia 3.X sa dočkala čiastočnej opravy a *overflow:scroll* je použiteľné, ale funkčné rolovanie i s *overflow: auto;* vo vnútri prvkov je až v Android 4.X.

Ako univerzálnejšie riešenie navrhujem použiť malý trik ktorý bude univerzálne fungovať i na starších zariadeniach. Rolovanie na bežných internetových stránkach funguje na všetkých zariadeniach. Dôvodom je, že na týchto stránkach dochádza k rolovaniu tela (prvok „body“) v rámci stránok. A dokonca je rolovaná zotrvačne. Má to však obmedzenia. Na jednej stránke/obrazovke môže byť iba jedna rolovacia oblasť a všetky ostatné elementy majú pomocou CSS nastavenú vlastnosť *position:fixed*, čím sú zafixované na presnej pozícií a zvyšok stránky sa roluje. Na niektorých zariadeniach ani tento trik nefunguje a dochádza

[7] https://developer.apple.com/support/appstore/

k poskakovaniu alebo preblikávaniu elementov - najčastejšie kvôli výkonnostným problémom. Tento problém by sa dal vyriešiť použitím *iFrame* rámcov. Jeden iFrame by zodpovedal iba rolovacej časti a je umiestnený na potrebnej pozícií obrazovky. Avšak v súčasných web-kitoch (iOS i Android) na mnohých zariadeniach rolovanie vo vnútri iFramu vôbec nefunguje. Tvrdenia v tomto odstavci sú vyvodené na základe testov v kapitole 6.3.

Na vyriešenie rolovania na väčšine súčasných zariadení je možné použiť i externé *polyfill*[8] javascriptové riešenie - napríklad *Overthrow*[9] z dielne Filament Group alebo *iScroll 5*[10]. Ďalším problémom je počet objektov v rolovacom elemente. Ak aplikácia umožňuje prehliadať niekoľko tisíc objektov, napríklad máp, nieje potrebné ich zobraziť všetky - čím viac prvkov, tým pomalšia aplikácia. Môj návrh riešenia je postavený na tom, že pri rolovaní stačí načítať to, čo používateľ aktuálne vidí. Na druhom mieste to, čo pravdepodobne bude chcieť vidieť. V prípade, že bude mať záujem o ďalšie objekty, tak tie sa budú postupne načítavať. Plynulosť rolovania bude závisieť na rýchlosti získavania zobrazovaných dát, preto je vhodné zvoliť vyhovujúce množstvo aktuálne zobrazených dát. Základom návrhu je časovač, ktorý priebežne v pravidelnom intervale kontroluje ako veľmi je odrolovaný zoznam objektov. Ak dosiahne požadovanú hranicu, začnú sa načítavať a zobrazovať ďalšie objekty. Hranica môže byť napríklad odrolovanie siedmych osmín rolovacieho prvku, ktoré používam v demonštračnej aplikácii. Takto môže fungovať v podstate nekonečné rolovanie. Rolovanie sa dá vylepšiť ďalšími rôznymi úpravami. Príkladom úprav môže byť odstraňovanie dostatočne odrolovaných objektov alebo vynechanie načítavania v prípade veľmi rýchleho rolovania.

V demonštračnej aplikácií sú všetky zoznamy výsledkov zotrvačne rolovateľné. Využíva sa varianta *-webkit-overflow-scrolling: touch*, ktorá umožňuje rolovanie na najnovších zariadeniach, v ostatných prípadoch sa používa Overthrow Polyfill riešenie. Bližší popis sa nachádza v popise realizácie aplikácie. Pri postupnom načítavaní sa načíta približne tridsať výsledkov. Ďalšie výsledky sa načítavajú po dosiahnutí siedmich osmín odrolovaného obsahu.

Prechody medzi obrazovkami aplikácie

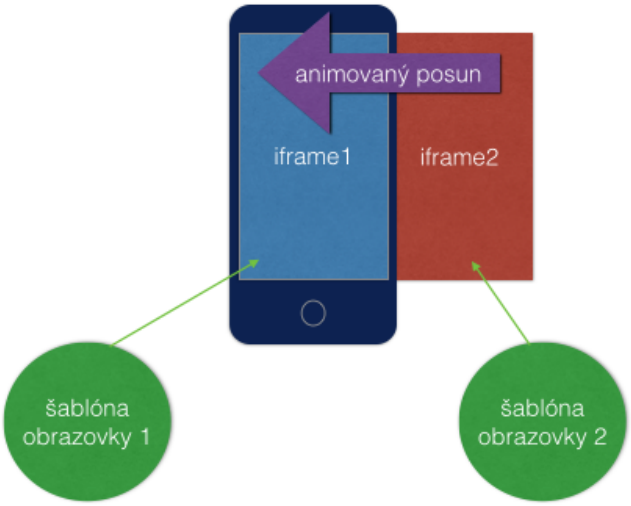
Prepínanie obrazoviek je okamžik, kedy sa všetok zobrazovaný obsah nahradí iným. Pri dátovo náročnom obsahu vzniká v používateľskom rozhraní pocit pomalých reakcii a trhania aplikácie. V bežných webových prezentáciach sa to deje prekreslením obrazovky. Natívne mobilné aplikácie však uprednostňujú animácie a určitú hierarchiu obrazoviek.

Návrh ako zrýchliť túto operáciu najlepšie popíšem na základe obrázku 5.1. Základom sú 2 rámce pre zobrazenie obrazovky aplikácie (*iframe1* a *iframe2*). V jednom rámci používateľ vidí aktuálnu obrazovku. Často z aktuálnej obrazovky je možné prejsť na malé množstvo (1-2) iných pravdepodobných stránok. Táto metóda umožňuje prednačítanie skrytých stránok. Dá sa predpokladať, že používateľ po zobrazení určitej stránky, strávi chvíľu prezeraním obsahu. Tento čas sa dá využiť na načítanie obsahu druhej obrazovky. Vo chvíli kedy používateľ sa presunie na nasledujúcu obrazovku bude obsah pripravený. Samozrejme túto metódu ide používať i bez prednačítavania. Mnoho času získa i samotná animácia prepnutia obrazoviek (za predpokladu hardvérových akcelerácií). Počas animácie môže prebiehať načítavanie čiastočne skryté pred zrakom používateľa. Animácia čiastočne odpúta jeho pozornosť. Tento

[8] Polyfill je prídavný kód, ktorý napodobňuje budúce API poskytovaním núdzovej funkcie na starších webových prehliadačoch. Voľný preklad z definície Paula Irisha - http://www.paulirish.com/i/7570.png

[9] https://github.com/filamentgroup/Overthrow

[10] http://cubiq.org/iscroll-5



Obrázek 5.1: Návrh urýchlenia prepínania obrazoviek

návrh pôjde uplatniť i na všetky ostatné prechody obrazoviek.

Špeciálnym prípadom môžu byť dlhé zoznamy klikateľných odkazov, pomocou ktorých sa zobrazí odlišný obsah. Nielen že nejde využiť prednačítavanie, ale ešte tento prípad vyžaduje množstvo obrazoviek. Navrhujem použiť jednu obrazovku, ktorej obsah sa nastaví počas animácie zobrazenia. Keď sa jedná o zoznam podobných elementov je vhodné nenútiť používateľa sa vrátiť o krok späť, ale dať mu možnosť prepnúť sa na nasledujúci alebo predchádzajúci objekt/obrazovku priamo. Vďaka tejto možnosti sa zredukuje počet možností ďalších obrazoviek na dve. Navyše používateľ bude mať väčšiu slobodu ako dosiahnuť požadovaný cieľ. Predchádzajúcu a nasledujúcu obrazovku už je možné prednačítať. Zároveň vždy jedna obrazovka je zobrazená, druhá je nasledujúca a tretia je predchádzajúca. Ak sa na niektorú z nich presunieme, vždy nám ostanú dve ďalšie. Ktoré môžme použiť na prednačítavanie ďalšej stránky.

Napríklad máme tri obrazovky - zobrazenú, predchádzajúcu a nasledujúcu. Ak sa používateľ sa prepne na nasledujúcu obrazovku, tá sa stane aktívnou. Predchádzajúca aktívna sa odsunie na pozíciu predchádzajúcej. Tretia obrazovka sa presunie na pozíciu nasledujúcej a aktualizuje svoj obsah. Ekvivalentne to funguje i do druhej strany.

Na základe myšlienok týchto návrhov sa dá riešiť mnoho podobných situácii.

V rámci aplikácie sa využívajú popísané princípy, pričom niektoré obrazovky sú na ľavej strane iné na pravej. Skrývajú sa tiež na zodpovedajúce strany aby používateľ nadobudol pocit, že sa posúva po obrazovkách hlbšie a hlbšie do aplikácie smerom zľava doprava. Rotačný princíp je demonštrovaný v rámci obrazoviek s detailami mapy.

Ťahacie gestá

Ťahacie gesto s prichytávaním (*snap* gesto) je zaužívané hlavne v natívnych mobilných aplikáciach ako i operačných systémoch. V iOS, Android i Windows Phone sa týmto gestom prepínajú jednotlivé plochy, alebo vysúvajú postranné menu. Preto je potrebné preniesť tento ovládací prvok i do webových mobilných aplikáciach.

Existuje niekoľko riešení, ktoré je možné uplatniť priamo v aplikácií. Existujúce riešenia sú popísané v poslednom odstavci kapitoly 2.2.

Je možné si vytvoriť podporu týchto gést i v javascripte. Dôvodom môže byť použitie preferovanej knižnice alebo obmena funkčnosti gesta. Napr. rozdielne je správanie vytiahnutia nového prvku ponad starý, alebo súčasné presunutie nového a odsunutie starého. Tento návrh je zobecnením implementácie frameworku *SnapJS*(2.2). Pre tvorbu ťahacieho gesta je možné pracovať s dotykovými udalosťami, zachytávaných na ťahanom HTML prvku. K dispozícií sú podľa špecifikácie štyri dotykové události[11]:

- *touchstart* - priloženie prstu na displej
- *touchmove* - pohybovanie prstom položeným na displeji
- *touchend* - zdvihnutie prstu z displeju
- *touchcancel* - prerušenie gesta

Udalosť *touchstart* nám inicializuje gesto, a spustí odchytávanie *touchmove* udalostí. Reakciou na udalosť pohybu bude posun požadovaných prvkov (ťahaného prvku, prípadne odsúvaných prvkov). Musí sa zaznamenávať vzdialenosť prstu od počiatočného dotyku. Táto vzdialenosť by sa mala merať iba v ose pohybu. Je vhodné obmedziť i smer posuvu kruhovým výsekom, aby bolo možné detekovať horizontálne, vertikálne prípadne iné snap gestá. SnapJS detekuje uhol pohybu 45 stupňov na jednu stranu osi a 45 stupňov na druhú. O nameranú vzdialenosť sa súčasne s udalosťami *touchmove* posúva ťahaný prvok. Je nutné použiť hardvérové akcelerácie popísané v kapitole 4.1 aby animácia bežala čo najviac oddelene od réžie gesta. Na udalosť *touchend* alebo *touchcancel* zareaguje rozhodovací algoritmus, ktorý môže označiť gesto za neúspešné a animáciou vráti prvky na pôvodnú polohu. Ak je gesto označené za úspešné, animáciou sa presunie ťahaný prvok na cieľovú pozíciu[12]. Rozhodovacia podmienka môže byť ľubovolná. Napríklad posun o určitú vzdialenosť, viditeľnosť určitých percent prvku na obrazovke a podobne.

V ďalších verziách demonštračnej aplikácie je plánované prepínanie detailov máp pomocou týchto gést.

[11] http://www.w3.org/TR/touch-events/
[12] Z toho názov *snap*.

Kapitola 6

Demonštračná aplikácia a testovanie

Demonštračná aplikácia je mobilná webová aplikácia zameraná na najnovšie mobilné operačné systémy. Je optimalizovaná a testovaná na systéme iOS[1] a posledných verziách Androidu[2]. Aplikácia má slúžiť na vyhľadávanie a atraktívne zobrazovanie historických máp, ktoré sú k dispozícií na serveroch svetových i českých inštitúcií. Aplikácia demonštruje nový, touto prácou navrhnutý, prístup k tvorbe multiplatformových mobilných webových aplikácií.

V následujúcich podkapitolách je popísaný návrh, implementácia a testovanie tejto aplikácie. Prvá podkapitola 6.1 obsahuj analýzu významu aplikácie a cieľovej skupiny používateľov a návrh aplikácie vrátane diagramov a návrhov obrazoviek aplikácie. V podkapitole 6.2 sú popísané najzaujímavejšie konkrétne časti realizácie aplikácie. Posledná a najrozsiahlejšia kapitola 6.3 je o testovaní postupov, ktoré boli navrhnuté v rámci tejto práce a realizované v rámci demonštračnej aplikácie.

6.1 Analýza a návrh aplikácie

Aplikácia pre prácu s historickými mapami patrí medzi aplikácie, kde prevažná časť aplikácie je závislá na dátach zo serveru - mapách. Máp je veľké množstvo a preto nemôžu byť uložené lokálne. Ak by sa uvažovalo o natívnej aplikácií, tak by väčšina aplikácie bežala vo webovom rozhraní. Natívne mapové rozhranie na mobilných platformách nie je schopné pracovať s dostupnými historickými mapami. Z tohto dôvodu je vhodné vytvoriť webovú aplikáciu, zaobaliť ju do natívnej aplikácie a ako pridanú hodnotu získame väčšiu slobodu pre prenos aplikácie na rôzne platformy.

Analýza významu a prínosu mobilnej aplikácie pre staré mapy

Už slovo mobilná poukazuje na najväčší prínos - mobilitu. Použitie v teréne. Vďaka moderným telefónom a tabletom, ktoré už takmer všetky obsahujú GPS lokátor, multidotykový displej a pripojenie k internetu, by bola rýchlosť vyhľadávania a komfort používania prispôsobenej mobilnej aplikácie výraznou zmenou oproti rozhraniu v internetovom prehliadači.

[1] Konkrétne sa jedná o verziu iOS 6 a 7
[2] Konkrétne sa jedná o verziu 4.4.X

Nemalým prínosom je zachovávanie kultúrneho dedičstva, ktoré každým zviditeľnením má väčšiu šancu na zachovanie pre ďalšie generácie.

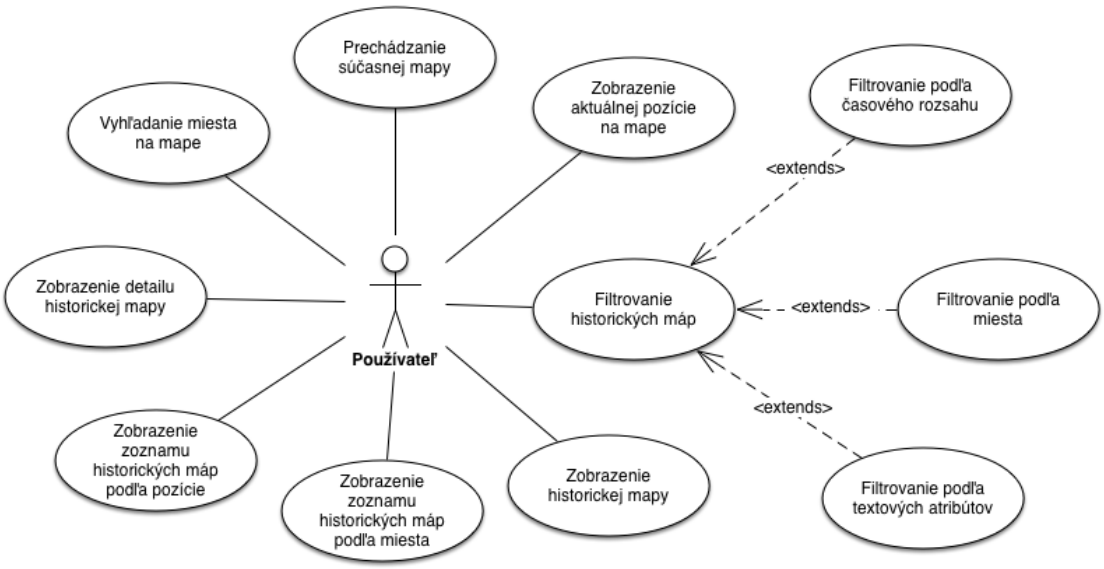
Cieľová skupina používateľov aplikácie

Aplikáciu využijú ako vedeckí pracovníci, ktorí v teréne môžu sledovať časový vývoj územia, tak i najväčšia cieľová skupina, bežní používatelia, ktorých jednoducho zaujíma, čo sa niekedy nachádzalo na mieste, kde práve stoja. Z vedeckých odvetví s mapami pracujú kartografovia, etnológovia, geológovia, ornitológovia, archeológovia, historici a ďalší odborníci, ktorí kombinujú historické a geografické údaje.

Vlastnosti a obrazovky aplikácie

Mobilná aplikácia by mala v prvom rade umožňovať prehliadanie dát. Pridávanie dát a georeferencovanie by bolo, kvôli potrebe pomerne veľkej plochy displeja, na mobilnom zariadení nepraktické. Napriek tomu, že aplikácia je v podstate mobilnou aplikáciou nad *oldmaponline.org*, prínosom by malo byť lepšie vyhľadávanie a filtrovanie máp na základe ostatných atribútov. S mapou by malo byť možné v reálnom čase pohodlne pracovať, prehliadať detaily a zachovať základné vlastnosti interaktívnych máp. Zároveň by mala zobrazovať vo vizuálne príjemnej forme informácie o konkrétnych mapách. Popri zobrazovaní máp na konkrétnych pozíciach by malo byť možné vyhľadať mapy i na miestach zadaných používateľom. Z internetovej verzie by bolo ideálne zachovať možnosti porovnania historickej mapy so súčasnou.

Návrh funkčnosti je popísaný prostredníctvom diagramu prípadov použitia na obrázku 6.1.



Obrázek 6.1: Use Case diagram - navrhované prípady použitia

Ako som už spomenul, základom aplikácie je vyhľadávanie relevantných máp medzi veľkým množstvom máp v katalógu. Preto na hlavnej obrazovke musí používateľ okamžite byť schopný odfiltrovať nepožadované výsledky. Na hlavnej stránke musia byť zobrazené

náhľady najrelevantnejších výsledkov a musia sa aktualizovať v reálnom čase. To bude poskytovať používateľovi spätnú väzbu a zároveň umožní presnejšiu interakciu s vyhľadávacími nástrojmi a zároveň zvyšuje šancu nájsť to, čo používateľ požaduje. Jednotlivé parametre vyhľadávania a spôsoby realizácie usporiadané podľa priority sú v nasledujúcom zozname:

1. **Vyhľadávanie podľa geografického umiestnenia** - vyhľadávanie funguje na základe porovnania so súčasnou mapou. Používateľ bude na väčšine obrazovky vidieť súčasnú mapu, prostredníctvom ktorej si dokáže známymi spôsobmi interakcie z iných mapových aplikácií zobraziť požadovanú oblasť. Primárne bude zobrazovať aktuálnu GPS pozíciu, ak bude k dispozícií. Na mape bude pozícia používateľa zobrazená „špendlíkom“. Pri presunutí sa mimo túto pozíciu, bude mať používateľ možnosť sa vždy na svoju GPS pozíciu vrátiť. Ak GPS súradnice k dispozícií nebudú, tak pri spustení primárne bude na hlavnej obrazovke zobrazený celý svet.

2. **Vyhľadávanie podľa názvu miesta** - jedná sa o klasické vyhľadávacie pole, kde používateľ zadá text - miesto, ktoré chce na mape vyhľadať. Dané miesto sa vyznačí „špendlíkom“, ďalej je možné pracovať s mapou ako v bode 1. alebo kliknúť na „špendlík“ a vyhľadať mapy, na ktorých sa dané miesto nachádza.

3. **Filtrovanie pomocou časového rozmedzia** - pri vyhľadávaní historickej mapy väčšinou nepoznáme presný rok, ale väčšinou poznáme nejaké obdobie, z ktorého mapy chceme zobraziť. No zároveň by používateľské rozhranie používateľa o možnosť presnej voľby rokov pripraviť nemalo. K zabezpečeniu tejto funkčnosti použijeme objekt, ktorý budeme nazývať časová osa, na ktorej bude možné zvoliť rozsah rokov, v ktorom sa vyhľadávané mapy musia nachádzať.

4. **Atribúty** - textové vyhľadávanie v metadátach - bude umožňovať filtrovanie pomocou atribútov. Pri potvrdení sa výsledky vyhľadávania aktualizujú, aby boli relevantné. Jedná sa konkrétne o tieto atribúty:

   - obsiahnutý text v popise
   - názov
   - autor
   - vydavateľ
   - mierka mapy

5. **Manuálne prechádzanie výsledkov vyhľadávania** - poslednou možnosťou bude nízkoúrovňové vyhľadávanie pomocou manuálneho prechádzania jednotlivých výsledkov v rolovacom zozname.

Užívateľské rozhranie aplikácie má byť zamerané na čo najrýchlejšie a najpríjemnejšie vyhľadanie potrebných dát.

Ukážky návrhu, rozloženia a dizajnu obrazoviek nájdeme na obrázku 6.2. Jedná sa o 6 obrazoviek. Úlohou úvodnej obrazovky je zdržať používateľa, kým sa podarí načítať všetky potrebné časti aplikácie. Po zobrazení tlačítka je možné sa dotknúť kdekoľvek na obrazovke pre presun na ďalšou obrazovku. Základom je hlavná obrazovka, na ktorej bude možné nastaviť všetky parametre vyhľadávania, a zobraziť výsledky. Vyhľadávanie relevantných dát je **hlavnou funkciou** aplikácie. Práca s konkrétnymi historickými mapami je záležitosťou práce s detailom mapy. Preto i pri návrhu som sa zameral na tieto 2 časti používateľského

Obrázek 6.2: Návrhy obrazoviek na mobilných telefónoch

rozhrania. Do úvahy pripadala i varianta, kde by sme presunuli vyhľadávanie na inú obrazovku. Avšak vzhľadom k tomu, že i prechádzanie mapy je vyhľadávanie, tak by nebolo ideálnym riešením jednotlivé prvky pre vyhľadávanie rozdeliť na rôzne obrazovky. Na opačnej strane bolo dôležité nechať obrazovku dostatočne priehľadnú, teda všetky ďalšie súčasti aplikácie sú na iných obrazovkách. Historickú mapu musíme zobraziť na celej obrazovke - displej je pomerne malý, takže ho musíme využiť na maximum. Detailné informácie o mape sú tým pádom odsunuté na ďalšiu obrazovku. Dôsledkom je, že detail mapy sa skladá na mobilnom telefóne z dvoch obrazoviek:

- Náhľad mapy s príslušnými informáciami
- Samotná historická mapa cez celú obrazovku

Stránka s detailami mapy zobrazuje metadáta - informácie o mape, autorovi, vydavateľstve a podobne. Ďalej obsahuje náhľad mapy a tlačítko na zobrazenie kompletnej georeferencovanej mapy ovládateľnej prostredníctvom obrazovky mobilného telefónu. V prípade, že mapa nie je prispôsobená pre mobilné telefóny (mapy v nekompatibilných formátoch) bude možnosť ju otvoriť v internetovom prehliadači.

Ako vidno na obrázku 6.3, u tabletu sú obrazovky pre vyhľadávanie a prechádzanie listu výsledkov zlúčené. Rovnako sú zjednotené informácie o mape so samotnou historickou mapou.

Interakcia s mapou, časovou osou a zmeny atribútov sú najdôležitejšie časti. Interakcia so samotnou mapou je pri dnešných nástrojoch triviálna. Problém však nastáva vtedy, ak je potrebné počas interakcie s mapou vykonávať nejakú činnosť. Je potrebné zabezpečiť, aby sa všetky činnosti vykonávali mimo interakcie s mapou. Podobný problém sa vyskytuje u časovej osi, kedy používateľ môže prejsť v zlomku niekoľkých sekúnd stovky rôznych letopočtov. Je nutné zabezpečiť aby sa nenačítavali nepotrebné výsledky. U zmeny atribútov vyhľadávania je interakciou písanie textu alebo zmena mierky mapy, problém je rovnaký.

Stavebné bloky aplikácie

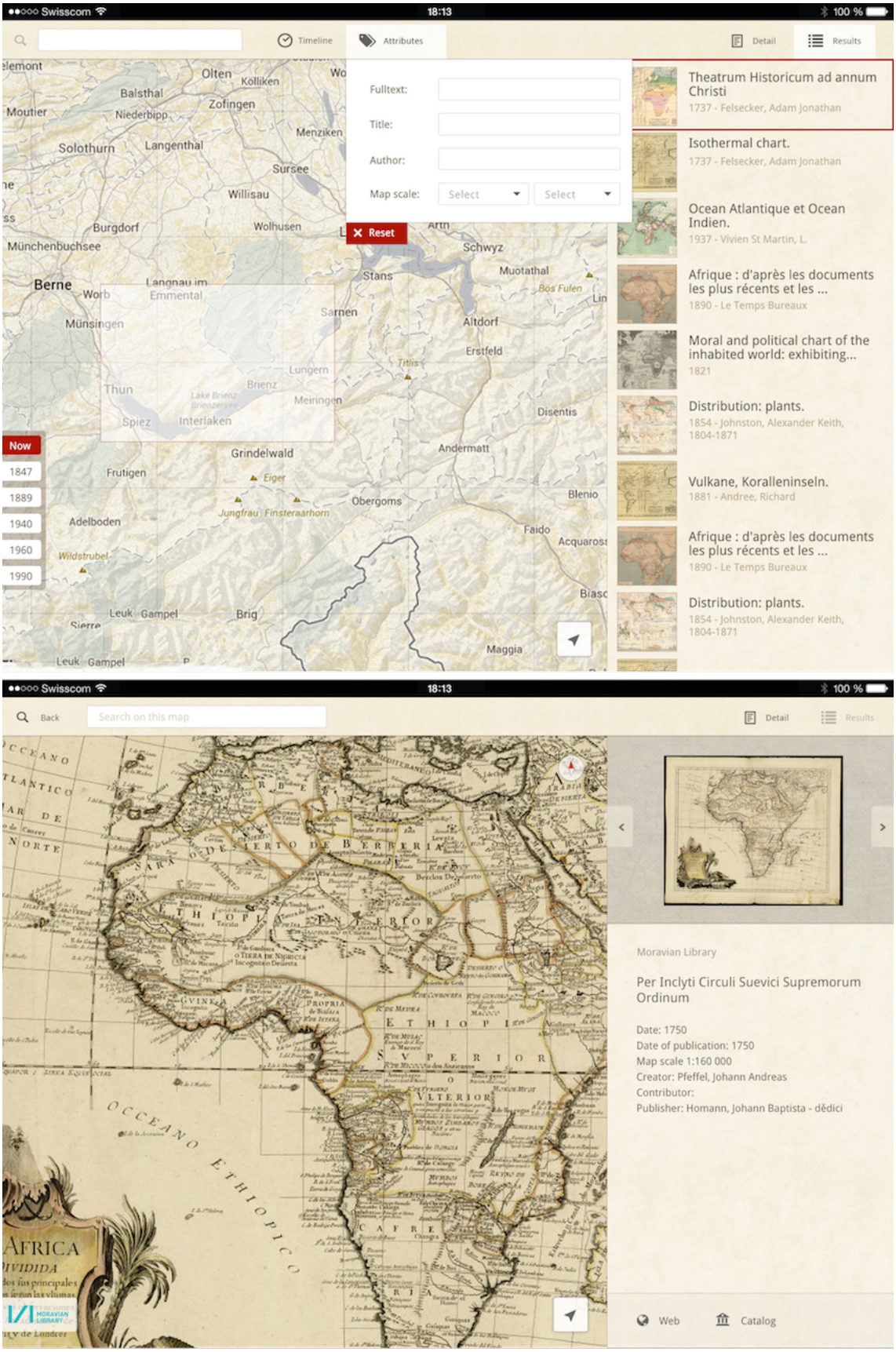
Princípy aplikácie sú založené na návrhoch riešenia jednotlivých nedostatkov mobilných aplikácií popísaných v tejto práci. Pre zjednodušenie uvádzam navrhnutý diagram prepojenia jednotlivých prístupov na obrázku 6.4. Bloky červenej farby znázorňujú obsah kapitoly o komunikačných kanáloch - 3. Zelená patrí kapitole zameranej na výkon - 4. Modrá farba sa týka používateľského rozhrania a zodpovedá kapitole 5.

6.2 Realizácia aplikácie

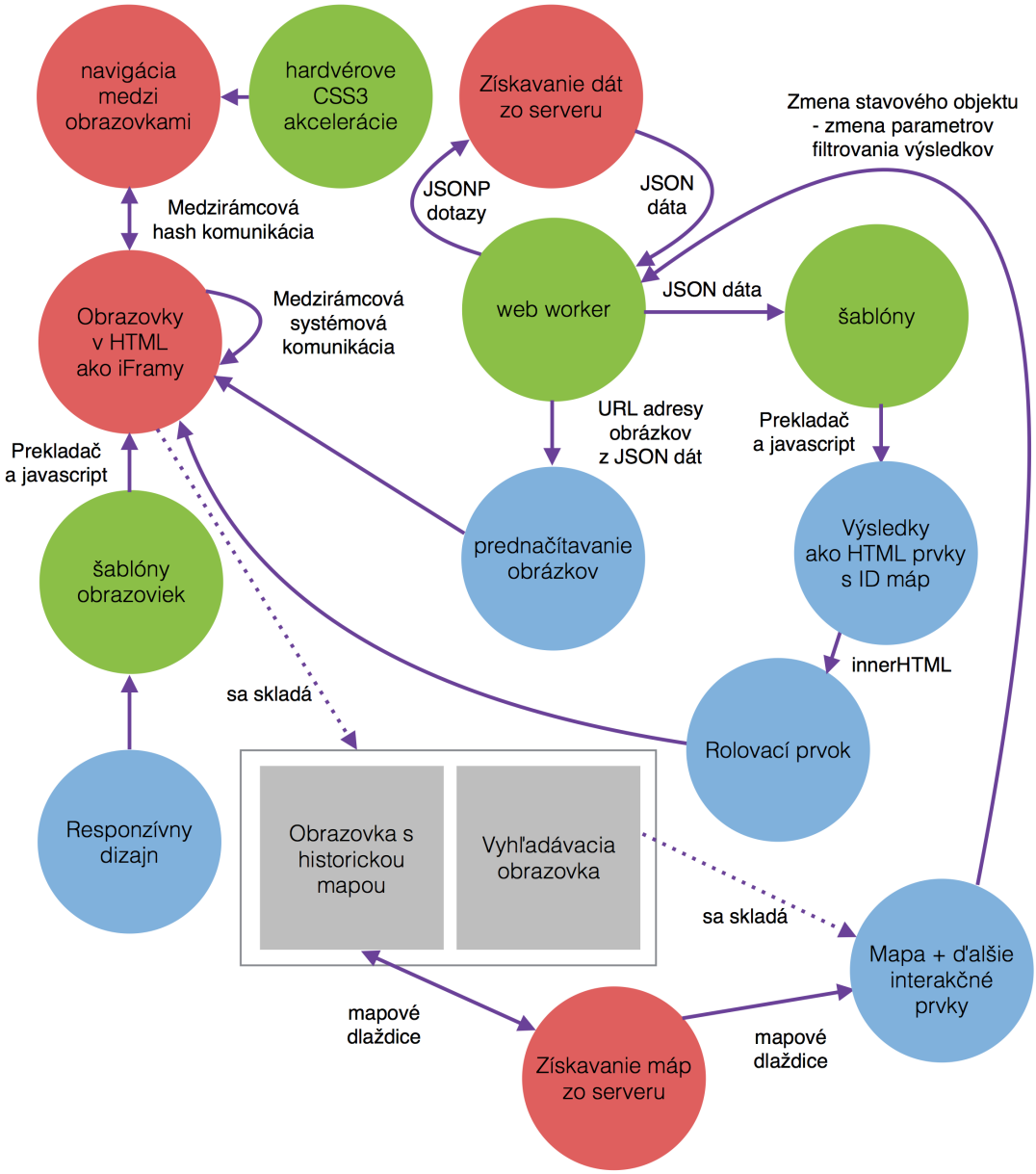
Táto kapitola popisuje konkrétny spôsob realizácie najzaujímavejších stavebných blokov aplikácie. V prvej časti sa venujem voľbe konkrétnych vývojových nástrojov a v ďalších častiach je rozbor implementácie interakcie s mapou, získavania dát zo serveru, prechádzania detailov a popis implementácie časovej osi.

Voľba vývojových nástrojov a technológií

Na realizáciu som zvolil framework PhoneGap, ktorý umožňuje vyvíjať HTML5/CSS3 aplikáciu súčasne pre iOS i Android. Zaobaluje web-kit do natívnej aplikácie, umožňuje prístup k API zariadenia, má podporu pre rôzne ikony a štartovacie obrazovky. Zároveň umožňuje použitie spoločne s Google Closure Tools, javascriptovým frameworkom, ktorý som zvolil



Obrázek 6.3: Návrhy obrazoviek na tabletoch



Obrázek 6.4: Návrh prepojenia blokov popísaných v práci pre vytvorenie úplnej demonštračnej aplikácie.

hlavne z dôvodu možnosti kompilácie. Kompilovaný javascript je úspornejší, optimalizovaný, s typovou kontrolou a rýchlejší. Mapový framework pre mapu na vyhľadávacej stránke i stránke so starou mapou bude Open Layers 3. Umožňuje používať viacero vrstiev, rôzne mapové zdroje a je prispôsobený pre mobilné zariadenia.

Interakcia s mapou

Realizovanou aplikáciou je mapová aplikácia. Zvolený mapový framework. V navrhnutej aplikácii je potrebné pri manipulácii načítavať nové výsledky. Pre plynulú prácu, ako je popísané v kapitole 5.1, bude sa všetká činnosť prevádzať až po skončení manipulácie s mapou.Využívať sa bude na detekciu zmeny udalosť mapy - *changeproperty*, ktorá je vyvolaná pri akejkoľvek zmene vlastnosti mapy. Ako zakončujúca udalosť je *moveend*, ktorá je vyvolaná po skončení posunu i zmene priblíženia mapy. Pred začiatkom práce s mapou budú obvykle načítané predchádzajúce výsledky. Ak v tej chvíli používateľ začne používať mapu, detekuje sa začiatok interakcie a nastaví sa stavová premenná reprezentujúca interakciu. Vo chvíli, keď zdvihne prst z obrazovky sa odošle požiadavok o nové dáta s oneskorením 300ms a zruší sa nastavenie stavovej premennej. Ako realizovať komunikáciu so serverom je popísané v kapitole 3.1 a znázornenie na obrázku 3.1. Poslanie požiadavku na server je odsunuté do druhého vlákna pomocou web workeru, popísaného v kapitole 4.2. Po získaní odpovede zo serveru dostaneme JSON objekt s údajmi o získaných mapách. Tieto dáta sú spracovávané vo web workeri. Po dokončení spracovávania sa overí či je nastavená stavová premenná interakcie. Ak je, znamená to, že používateľ znovu pracuje s mapou a získané výsledky môžeme zahodiť. V opačnom prípade sa výsledky použijú ako parametre šablóny podľa kapitoly 4.4. Zo šablóny dostaneme iba HTML kód (konkrétne zoznam HTML prvkov *li*), zodpovedajúci novým prvkom, ktorý vložíme do stránky. Rovnaká stavová premenná sa používa i pre časovú os alebo atribúty.

Spracovanie odpovedí zo serveru

Táto časť je zameraná na konkrétnu prácu web workeru po tom, ako mu hlavné vlákno zašle parametre filtrov vo formáte *JSON*. Vo web workeri získame odpoveď zo serveru, ktorá vyzerá ako v nasledujúcom kóde:

```
[
  {
    "id":"4730.003",
    "bbox":[-100.964444,-56.846111,141.445556,71.662222],
    "institution":"rumsey",
    "scale":50000000.000000,
    "date_from":"1937",
    "date_to":"1937",
    "title":"Ocean Atlantique et Ocean Indien.",
    "publisher":"Vivien St Martin, L."
  },
  {
    "id":"11540169",
    "bbox":[-126.032800,-45.438600,155.729400,71.330300],
    "institution":"harvard",
    "scale":18000000.000000,
    "date_from":"1890",
    "date_to":"1890",
    "pubdate_from":"1890",
    "pubdate_to":"1890",
```

```
21      "title":"Afrique",
22      "creator":"Giffault, E",
23      "publisher":"Le Temps Bureaux",
24      "catalog":"http://hollis.harvard.edu/?itemid=|library/m/aleph|011540169"
            ,
25      "viewer":"http://nrs.harvard.edu/urn-3:FHCL:2250066?buttons=y"
26    },
27      ...
28  ]
```

Adresy na serveri sú uložené v adresárovej štruktúre „*/img/(rozmer)/(inštitúcia)/(id).jpg*“. Pre zobrazenie výsledkov sa používa rozmer 75 (pixelov) a pre náhľad v detaile 350 (pixelov). Obrázok s rozmermi 350 pixelov je používaný i ako podkladová vrstva mapy. Z uvedenej odpovedi zo serveru dokážeme poskladať adresy potrebných obrázkov, ktoré sa prednačítavajú už v osobitnom vlákne spôsobom popísaným v kapitole 5.3. V dobe písania práce nie je pripravený mapový server pre mobilnú aplikáciu. Z toho dôvodu sa v demonštračnej aplikácii využívajú existujúce mapové servery, taktiež nemajú nastavené príslušné CORS hlavičky potrebné na prednačítavanie obrázkov. Podpora prednačítavania je implementovaná a pripravená na nasadenie.

Časová os

Časová os je založená na dvoj-posuvníkovej komponente *goog.ui.twoThumbSlider* z Google Closure Tools. Doplnenými metódami sú značky časových os, ktoré sú vygenerované na percentuálnych pozíciach. Minimálna hodnota je 0, označovaná ako „Past“ - minulosť. Maximálna hodnota je aktuálny rok, ktorý sa nastaví vždy po spustení aplikácie. Pri interakcii sa nad posuvníkmi zobrazujú roky výberu. Aby nedochádzalo k trhaniu pri používaní je pri posunu posuvníka spustený časovač na jednu milisekundu. Ak sa hodnota do jednej milisekundy nezmení, tak sa vypíše nad posuvník príslušný rok.

Prechádzanie detailov

Prechádzanie detailov jednotlivých starých máp je riešené podľa kapitoly 5.5. Konkrétne sa jedná o rotačnú zmenu obrazoviek. V implementácii sú použité 3 identické obrazovky - *iFramy*. Majú nastavené rôzne triedy pre predchádzajúci detail, aktuálny detail a nasledujúci detail. Pri dokončení načítania predchádzajúcej a nasledujúcej obrazovky sa nastaví do aktuálneho detailu *listener* pre posunutie na predchádzajúci resp. nasledujúci detail. Pri realizácii posunu, klepnutím na šípku alebo gestom posunutia (v ďalšej verzii aplikácie), sa vymenia triedy jednotlivých prvkov. Na viditeľných obrazovkách je zapnutá animácia.

6.3 Testovanie aplikácie

Náplňou tejto časti je na meraniach ukázať význam niektorých popisovaných prístupov a technológií. Testy sú zamerané na časti, ktorých prínos nemusí byť úplne zrejmý alebo miera zlepšenia nie je úplne odhadnuteľná. K testom je použitá demonštračná aplikácia. Prvé testovanie je zamerané na význam hardvérových akcelerácií. V ďalšom teste sa jedná o experimenty správania rolovania na rôznych zariadeniach. V závere každej sekcie sa nachádza vyhodnotenie výsledkov.

Test hardvérových akcelerácií

Kaskádové štýly CSS3 nám umožňujú hardvérovo akcelerovať animácie ako je popísane v kapitole 4.1. Cieľom testovania je odhaliť vplyv použitia hardvérových akcelerácií na *FPS* (*Frames Per Second* - počet snímkov za sekundu). Test je zameraný na prechody medzi obrazovkami. Obrazovky aplikácie sa prepínajú pomocou animácie. Plynulosť animácie je merateľná práve pomocou FPS.

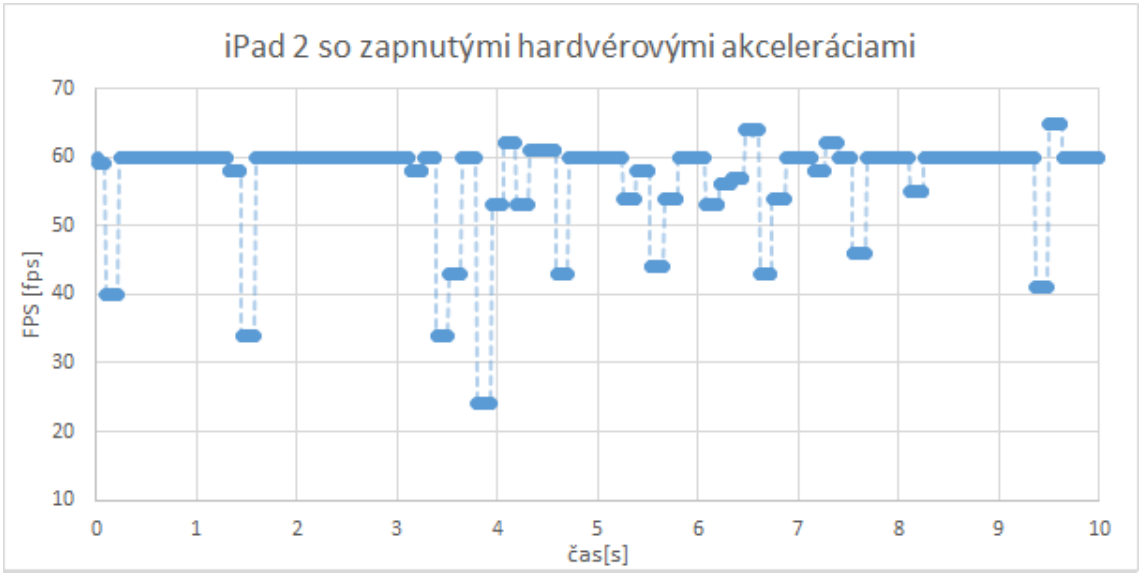
Tieto testy je nutné vykonávať na reálnych zariadeniach. Simulátory nedosahujú výkon odpovedajúci reálnym zariadeniam. Testované mobilné zariadenia budú všetky na platforme iOS, čo je primárna platforma pri vývoji demonštračnej aplikácie. Jedná sa o iPhone 4S, iPhone 5S a iPad 2 s operačným systémom iOS 7. Pre porovnanie jedno meranie je uskutočnené i na notebooku MacBook Pro s operačným systémom Mac OS X 10.9.4.

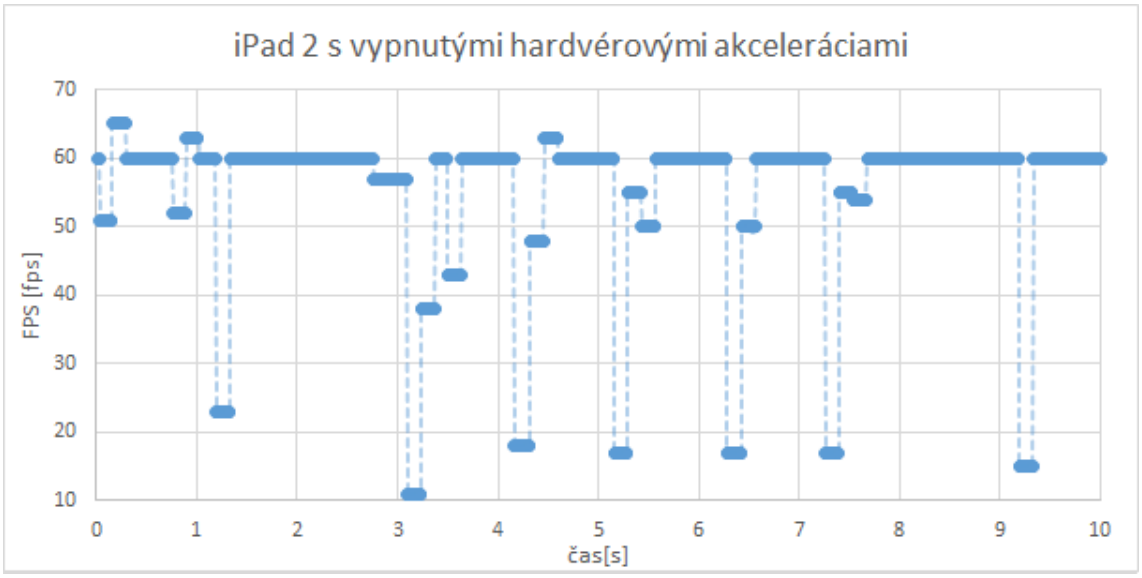
Samotné meranie prebieha na demonštračnej aplikácií s pomocou špeciálnej javascriptovej triedy. Aplikáciu je nutné preložiť so špeciálnymi parametrami, popísanými v prílohe B. V rámci aplikácie sa zobrazí monitor FPS v reálnom čase. Pri kliknutí na políčko s textom aktuálneho FPS spustí automatický test, ktorý prechádza jednotlivé obrazovky aplikácie a každých 10 milisekúnd zaznamenáva FPS. Test trvá 10 sekúnd na konci testu je zobrazené minimálne a priemerné FPS počas animácií. Čas začiatku animácie sa zaznamenáva ako čas akcie vyvolávajúcej prechod obrazoviek (napr. stlačenie tlačítka). Čas ukončenia udalosťou dokončenia animácie - *transitionend*. Po skončení testu sú do konzoly prehliadača vypísané výsledky. Výsledky obsahujú dvojice čas a FPS. K niektorým časom je doplnená poznámka udalosti - napr. začiatku a konca animácie.

Priebeh akcií testovacieho kódu je nasledujúci:

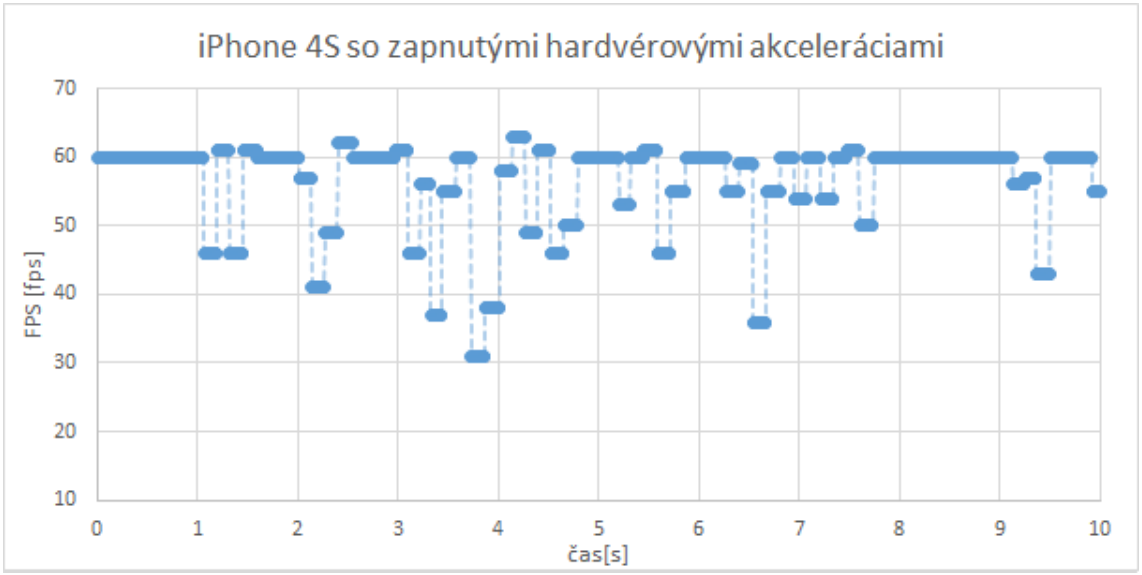
1. sekunda - Prechod na vyhľadávaciu obrazovku (z akejkoľvek obrazovky).
2. sekunda - Prechod na Obrazovku so zoznamom výsledkov (len mobilné telefóny).
3. sekunda - Klik na prvý výsledok nájdených starých máp - prechod na detail mapy.
4. sekunda - Prechod na nasledujúci výsledok.
5. sekunda - Prechod na ďalší nasledujúci výsledok.
6. sekunda - Prechod na ďalší nasledujúci výsledok.
7. sekunda - Prechod na predchádzajúci výsledok.
8. sekunda - Zobrazenie mapy (len mobilné telefóny).
9. sekunda - Prechod na vyhľadávaciu obrazovku.
10. sekunda - Vyhodnotenie testu a výpis výsledku.

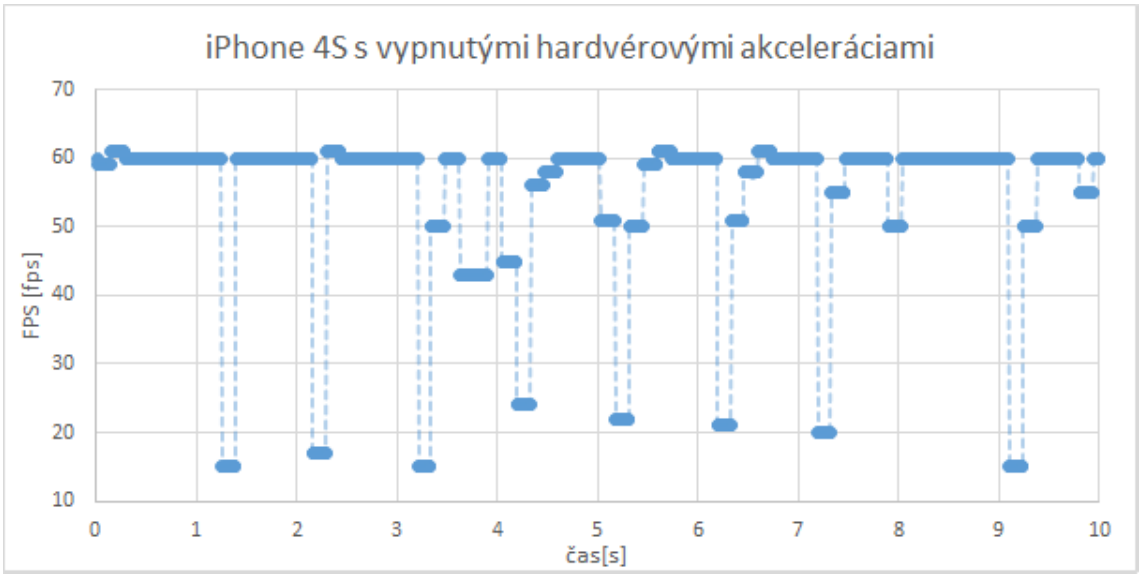
Základom testovania je vytvorenie dvoch verzií testovanej aplikácie. Jedna s použitím hardvérových CSS3 akcelerácií, druhá bez použitia. Oba príklady sú popísané v kapitole 4.1. Test bol vykonaný tak, že obe aplikácie boli postupne nainštalované na rovnaké zariadenie. Po spustení a načítaní aplikácie bol vykonaný test a uložené jeho výsledky. Test bol spustený vždy z uvítacej obrazovky a bol zopakovaný niekoľkokrát. Výsledky boli veľmi podobné. Rozdiely sú v kontexte s porovnávanými výsledkami dvoch aplikácií zanedbateľné. Výsledky z behov oboch aplikácií sú znázornené pomocou grafov na obrázkoch 6.5 až 6.8.



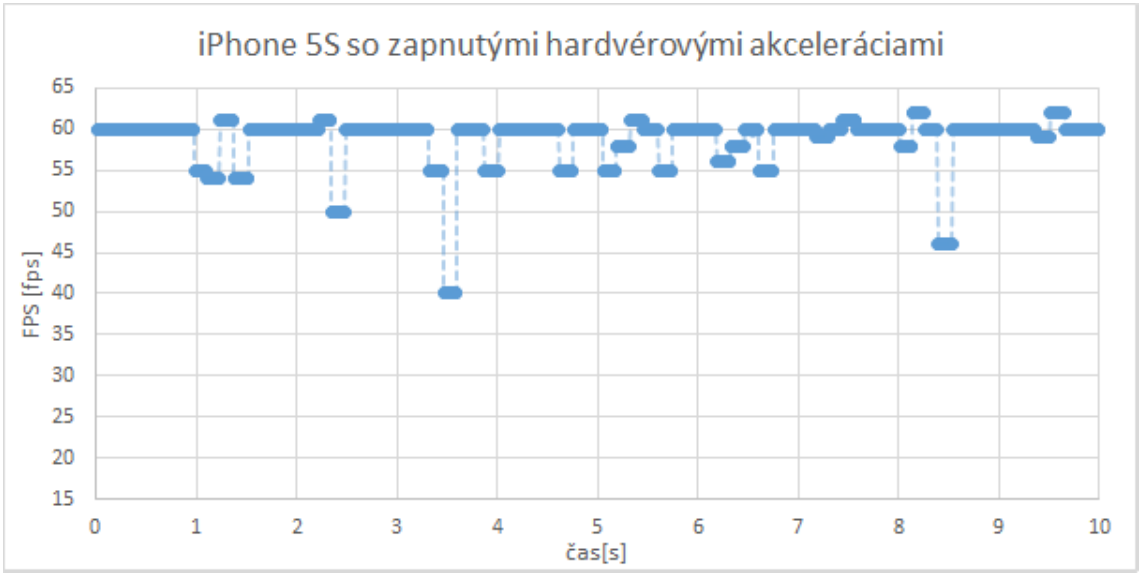


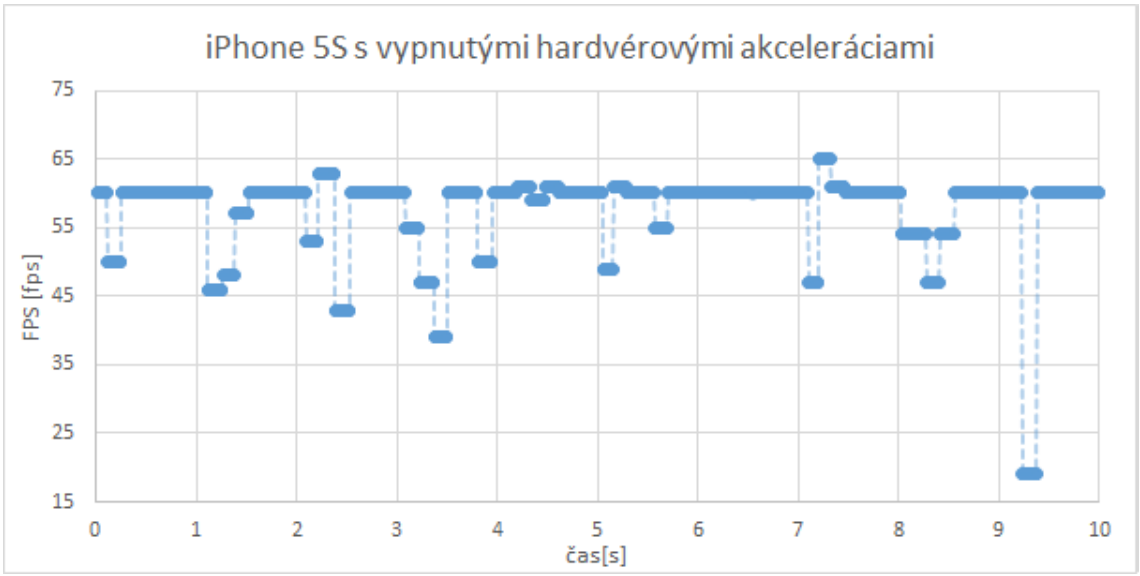
Obrázek 6.5: Porovnanie aplikácie s hardvérovými akceleráciami a bez nich na tablete



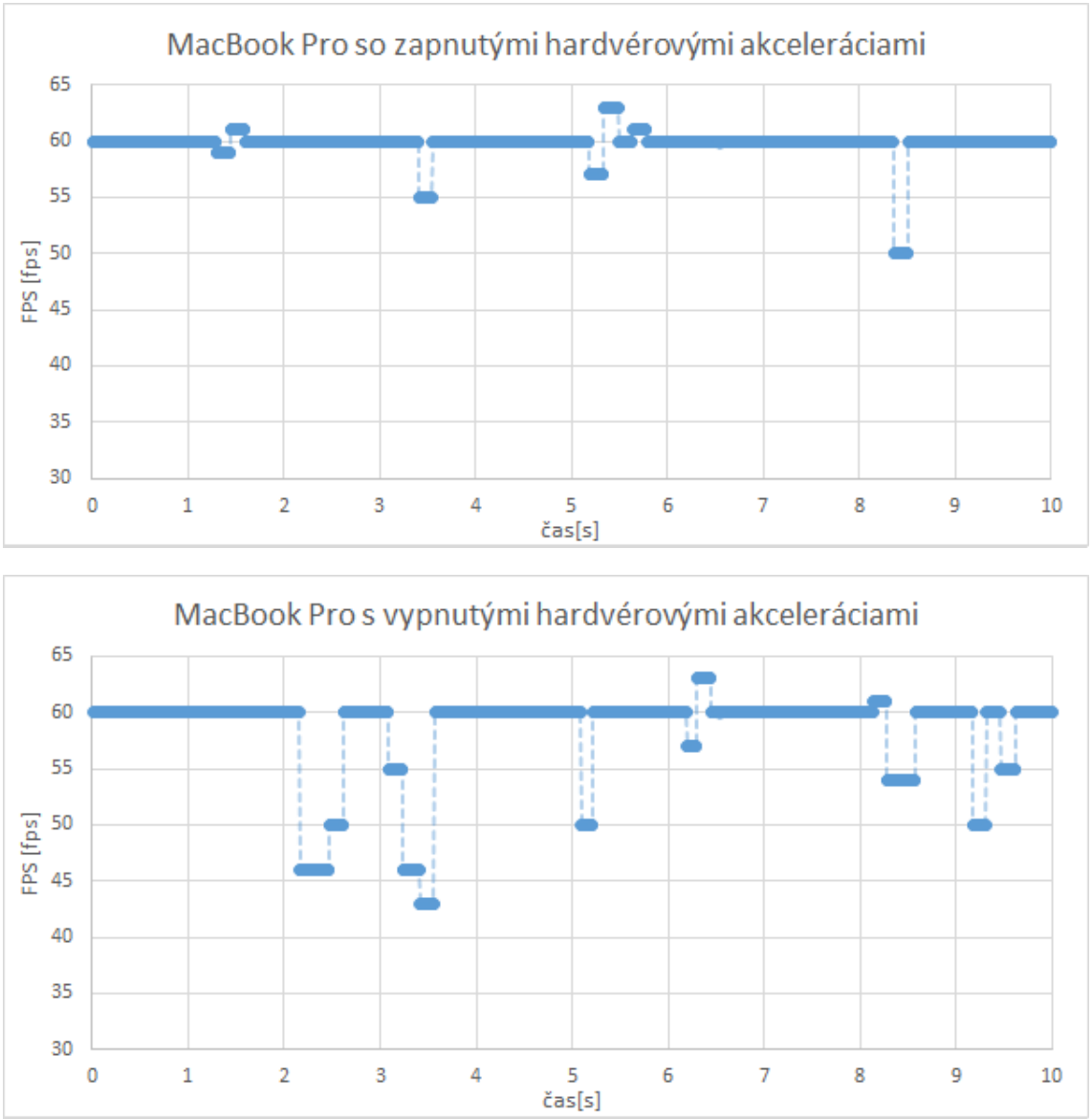


Obrázek 6.6: Porovnanie aplikácie s hardvérovými akceleráciami a bez nich na telefóne iPhone 4S





Obrázek 6.7: Porovnanie aplikácie s hardvérovými akceleráciami a bez nich na výkonnom telefóne iPhone 5S, kde sa rozdiely zmenšujú.



Obrázek 6.8: Pre demonštračné účely porovnanie aplikácie s hardvérovými akceleráciami na notebooku - ako vysokovýkonnom zariadení.

Prínos hardvérových akcelerácií pomocou CSS3 je najviditeľnejší na výsledných dátach získaných z testov na tablete iPad 2 (obr. 6.5) a telefóne iPhone 4S (obr. 6.6). Pri verzii s hardvérovými akceleráciami neklesá hodnota FPS pod 30[3]. Výnimkou je jeden prípad pri tablete iPad 2 - koniec tretej sekundy. Avšak pokles FPS je spôsobený načítavaním mapy až po animácií. Obrazovka je v tej chvíli statická a teda tento pokles nemá vplyv na plynulosť animácii. Animácie prebiehajú od celej sekundy a trvajú 400 ms. Smerodajný je vždy polsekundový interval po každej celej sekunde. Vo variante bez hardvérových akcelerácii práve v tomto intervale sa hodnoty FPS dostanú i pod 20 fps. Na výkonnejších zariadeniach ako iPhone 5S (obr. 6.7) alebo notebooku (obr. 6.8) vzhľadom k výkonu zariadenia nie sú rozdiely tak rapídne viditeľné a v oboch prípadoch sa väčšinou držia nad hranicou 30 fps. U iPhone 5S pri teste bez hardvérových akcelerácii klesne fps pod 30 iba pri návrate na vyhľadávaciu stránku. Prechod je zo stránky s historickou mapou. Obe obrazovky sú výkonovo náročné. Z uvedených dôvodov sa ukazuje použitie hardvérových akcelerácií vhodné pre zlepšenie výkonu mobilnej aplikácie. Najväčší význam použitia je u menej výkonných zariadení, kedy je možné časť náročných úloch preniesť na grafický čip.

Test prínosu osobitného vlákna pre asynchrónne dotazy

Webová technológia s názvom Web Worker umožňuje použitie viacerých vláken v rámci javascriptu a komunikáciu medzi nimi. Viac o technológii a jej použití sa nachádza v kapitole 4.2. V demonštračnej aplikácií sa v osobitnom vlákne vykonávajú dotazy na server. V rámci vlákna sa odošle JSONP dotaz. Prijatá odpoveď zo serveru je následne spracovaná. Najväčší význam má použitie osobitného vlákna práve kvôli spracovaniu výsledkov, ktoré je popísané v kapitole o realizácií 6.2.

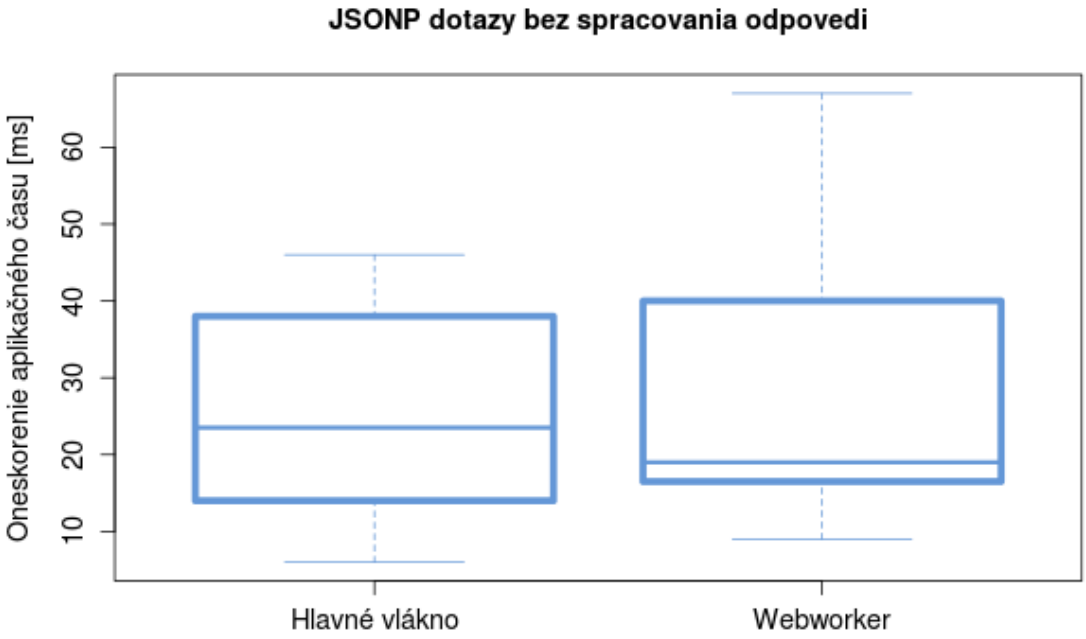
Pre testovanie sú vytvorené 2 verzie aplikácie. Test je automatický, zabezpečuje ho špeciálna trieda, ktorá riadi aplikáciu a zaznamenáva merané údaje. Prvá verzia testuje iba samotné asynchrónne dotazy (JSONP) a druhá testuje dotazy spolu s dodatočným spracovaním odpovedí. Pri meraní sa zaznamenávajú 2 hodnoty: trvanie podľa systémového času a trvanie podľa aplikačného času. Tieto hodnoty sú merané vždy v hlavnom vlákne. Začiatok vo chvíli keď sú nastavené parametre dotazu pripravené na zaslanie. Koniec je zaznamenaný až keď sú získané dáta pripravené v hlavnom vlákne. Aplikačný čas sa merá pomocou časovača, ktorý tiká každých 25 milisekúnd vo voľnom procesorovom čase hlavného vlákna. Hodnota, ktorú z týchto dvoch údajov vypočítavam je oneskorenie aplikačného času. Výpočet je jednoduchý, jedná sa o rozdiel systémového a aplikačného času (prenásobeného 25). Pri preklade aplikácie s danými parametrami (popísané v prílohe B) sa do demonštračnej aplikácie pridá políčko, v ktorom beží aplikačný čas. Pri interakcii s aplikáciou je možné sledovať ako sa tento čas spomaľuje alebo zrýchľuje v závislosti na náročnosti operácie. Pri kliknutí na políčko so systémovým časom sa spustí automatizovaný test, kedy sa opakovane zasiela rovnaký dotaz na server. Počet opakovaní je 100. Ďalší dotaz sa zasiela až po spracovaní predchádzajúceho s 3 sekundovým oneskorením. Dôvodom oneskorenia je, aby prekresľovanie používateľského rozhrania nemalo vplyv na test. Pred každým opakovaním je aplikačný čas vynulovaný. U systémového sa zaznamenáva vždy prírastok - doba trvania. Testy boli vykonané na mobilnom telefóne iPhone 5S.

Verzia, ktorá meria trvanie iba od samotného dotazu do doručenia odpovedí, je demonštračná. Má ukázať, že použitie asynchrónnych dotazov je s použitím i bez použitia web workerov ekvivalentné. Kvartilový graf[4] na obrázku 6.9, ukazuje že použitie web wor-

[3] V teste beriem hodnotu 30 fps ako hranicu pre plynulosť animácii.

[4] Kvartilový graf zobrazuje kvartíly dát ako okraje (spodný - 1. kvartíl a horný - 3. kvar-

kerov je mierne výhodnejšie ak sa zameriame na 50% stredných hodnôt. Ak sa zameriame na rozsah oneskorenia, vychádza lepšie verzia s hlavným vláknom. V každom prípade sa jedná o milisekundy. Pri teste priemerné oneskorenie v hlavnom vlákne bolo približne 26 ms a pri použití web workeru 25 milisekúnd. Web Workery by však mohli byť potenciálne ešte pomalšie, ak by sa medzi vláknami prenášal väčší objem dát[5].



Obrázek 6.9: Graf nameraného oneskorenia aplikačného času pri asynchrónnych dotazoch bez dodatočného spracovania odpovedí

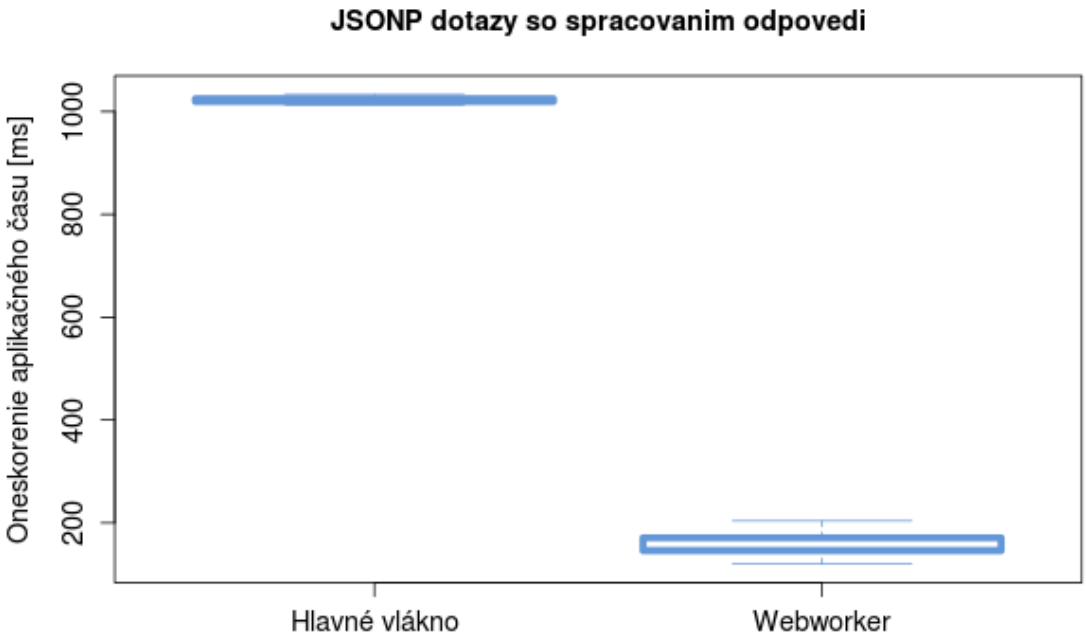
Druhá verzia testu meria trvanie i vrátane dodatočného spracovávania výsledkov. Dodatočné spracovanie v teste simulujeme prostredníctvom výpočtu, ktorý trvá 1 sekundu. Jedná sa o cyklus, ktorý počíta počet svojich behov a kontroluje čas. Ak dosiahne trvanie piatich sekúnd, ukončí sa. Výsledkom je porovnanie varianty s použitím web workeru a bez neho. Rozdiel je vidieť na grafe z obrázku 6.10. Rozdiel medzi oneskorením aplikačného času u spracovania v hlavnom vlákne a webworkerom je takmer 1 sekunda, ako by sa dalo predpokladkladať. Priemerné oneskorenie v hlavnom vlákne bolo približne 1024 ms a pri použití web workeru 176 milisekúnd.

Z uvedených výsledkov je zrejmé, že využitie web workerov pre asynchrónne dotazy na server nie je zbytočné, ak odpovede sú spracovávané v rámci web workeru. Z výsledkov sa dá vydedukovať, že použitie web workerov má význam už vo chvíli ak spracovanie výsledkov trvá aspoň niekoľko desiatok až stoviek milisekúnd.

---

tíl) strednej „krabicovej“ časti. V tejto časti sa nachádza približne 50% hodnôt. Čiara vo vnútri „krabicovej“ časti označuje medián hodnôt. Tenké čiary vedúce z tejto časti značia rozsah hodnôt. (http://www.mathwords.com/b/boxplot.htm)

[5]Tvrdenie je založené na výkonnostných testoch web workerov na odkaze spoločnosti Scott Logic: http://www.scottlogic.com/blog/2011/02/18/web-workers-part-1-performance.html



Obrázek 6.10: Graf nameraného oneskorenia aplikačného času pri asynchrónnych dotazoch s jednosekundovým spracovaním odpovedí

Test podpory rolovania na rôznych zariadeniach

Ako je popísané v kapitole 5.5 riešenie zotrvačného rolovania nie je úplne triviálne. Rozdiely sú medzi platformami i medzi jednotlivými verziami operačných systémov. Ako bude fungovať rolovanie na jednotlivých zariadeniach? Aká je najjednoduchšia varianta pre podporu najnovších zariadení. Aká varianta zabezpečí funkčnosť na väčšine používaných zariadení? Tieto otázky sú motiváciou pre tento test.

Pre testovanie je vytvorená špeciálna mobilná aplikácia. Tá obsahuje osem rolovateľných HTML elementov. Štyri horizontálne a štyri vertikálne. Z danej štvorice má každý element iný prístup v rolovaní obsahu. Konkrétne sa jedná o:

- Rolovanie tela *body* v rámci iFrame rámca.
- Rolovanie pomocou nastavenia CSS vlastnosti *overflow:auto*.
- Rolovanie pomocou nastavenia CSS vlastností *overflow:scroll* a špeciílnej vlastnosti pre webkit *webkit-overflow-scrolling:touch*.
- Rolovanie s použitím javasriptového riešenia *Owerthrow*[6] (viac v kapitole 5.5).

Táto aplikácia bola použitá pre testovanie podpory na jednotlivých zariadeniach. Použité boli reálne zariadenia i emulátory. Prehľad výsledkov testovania je vidieť v tabuľke na obrázku 6.11.

[6]https://github.com/filamentgroup/Overthrow

Rolovanie (zotrvačné)	iFrame <body>	overflow: auto;	webkit-overflow-scrolling: touch;	Overthrow polyfill
iOS 6 (iPhone 4)	NIE	NEZOTRVAČNE	ÁNO*	ÁNO
iOS 7 (iPhone 4S/5S, iPad 2/3)	NIE	NEZOTRVAČNE	ÁNO*	ÁNO
Android 2.3.x (Samsung Galaxy S2)	NIE	NIE	NIE	NEZOTRVAČNE
Android 4.0.x(Sony Xperia Sola)	ÁNO**	ÁNO	ÁNO	ÁNO
Android 4.1.x (Samsung Galaxy S3 Mini)	ÁNO**	ÁNO	ÁNO	ÁNO
Android 4.2.x (Motorola Moto X)	ÁNO**	ÁNO	ÁNO	ÁNO
Android 4.3.x (Samsung Galaxy Note 2)	ÁNO**	ÁNO	ÁNO	ÁNO
Android 4.4.x (Google Nexus 5)	ÁNO	ÁNO	ÁNO	ÁNO
Windows Phone 8 (Nokia Lumia 520)	ÁNO	ÁNO	ÁNO	ÁNO
	* pri niektorých spusteniach nefunguje vertikálne rolovanie ** v náhodných prípadoch po určitom čase prestalo rolovanie fungovať (pri prepnutí na vertikálne a späť na horizontálne rolovanie)			

Obrázek 6.11: Získané výsledky z testovania podpory rolovania na rôznych platformách

Zelenou označené možnosti, ktoré sa počas testovania prejavili ako spoľahlivé. Oranžovou farbou sú označené varianty, ktoré za určitých podmienok fungujú, ale nemusia. Napríklad v prípade iOS pomocou nastavenia CSS vlastností *overflow:scroll* a *webkit-overflow-scrolling:touch* rolovanie funguje ako má. Avšak v určitých chvíľach pri vertikálnej verzii začne rolovanie fungovať len v reakcii na horizontálne pohyby[7]. Ďalšou nepríjemnosťou sú chyby pri používaní iFrame rámcov. Aj keď u Androidu 4.0 - 4.3 funguje rolovanie tela iFrame rámca, nie je to žiadna záruka. V testovacej aplikácií skrytie horizontálnych prvkov, zobrazenie vertikálnych a následné vrátenie sa k horizontálnym prvkom zastavilo funkčnosť rolovania v iFrame rámcoch. Operácia zobrazovania a skrývania prvkov je pri webových mobilných aplikáciach často využívaná. Modrou farbou sú v tabuľke na obrázku 6.11 označené varianty, kedy rolovanie funguje, ale nie je zotrvačné. Červenou farbou sú vyplnené varianty, ktoré podporované nie sú.

Najvhodnejšou variantou sa javí použitie Overthrow javascriptového riešenia. Avšak sa jedná o riešenie javascriptom, teda v aplikácií bude bežať ďalší javascriptový kód len pre riešenie rolovania. Za zváženie stojí vynechať podporu starších zariadení alebo obetovať zotrvačnosť. Riešením môže byť i použitie odlišných prístupov na rôznych platformách. Použitie iFrame rámcov sa javí ako najmenej vhodná alternatíva.

[7]Používateľská diskusia na tému nesprávnej detekcie smeru rolovania (vrátane hypotéz dôvodov a riešení): http://travisjbeck.com/blog/browsers/webkit-overflow-scroll-touch-vertical-scrolling-only-responds-to-horizontal-swiping/

Kapitola 7

Záver

Hlavným cieľom tejto práce bolo navrhnúť nové postupy pre tvorbu mobilných webových aplikácií. Dôraz bol kladený na výkon a podobnosť používateľského rozhrania s natívnym. Pre demonštráciu návrhov a potreby testovania bola vytvorená aplikácia pre interakciu s historickými mapami.

Súčasťou testovania bolo porovnanie výsledkov s použitím navrhnutých postupov a bez nich. Testy merania rýchlosti prekresľovania počas animácií ukázali, že na vysoko výkonných zariadeniach sú hodnoty FPS (počet snímkov za sekundu) vyhovujúce v oboch prípadoch. Na menej výkonných zariadeniach boli hodnoty FPS pri použití navrhnutého postupu často 2-4krát vyššie. Ďalšie testy priniesli podobné výsledky v oblasti zasielania dotazov na server, kedy zlepšenie je významné v prípade dlhšieho dodatočného spracovania výsledkov.

Vytvorená demonštračná aplikácia v súčasnej verzii[1] umožňuje vyhľadávanie a zobrazovanie máp podľa používateľských kritérií. Súčasné omedzenie aplikácie je i v kvalite zobrazovaných historických máp z dôvodu nedoriešených zmlúv o použití mapových dát.

V ďalších verziách bude aplikácia obohatená o služby GPS a budú pridané možnosti prekrývania podkladovej a historickej mapy. V ďalších verziách je plánovaná možnosť používateľskej zmeny podkladovej mapy. Najnáročnejšia časť ďalšieho vývoja bude testovanie a optimalizácia aplikácií na čo najširšie spektrum mobilných platforiem a zariadení.

[1] Verzia 1.0

Referencie

[1] Avins, J.; Nguyen, J.: The Making of Fastbook: An HTML5 Love Story. 2012.
URL
http://www.sencha.com/blog/the-making-of-fastbook-an-html5-love-story

[2] Bidelman, E.: The Basics of Web Workers: The Problem: JavaScript Concurrency. 2010.
URL http://www.html5rocks.com/en/tutorials/workers/basics/

[3] Bidelman, E.: New Tricks in XMLHttpRequest2. 2011.
URL http://www.html5rocks.com/en/tutorials/file/xhr2/

[4] Bolin, M.: *Closure: The Definitive Guide*. Beijing: O'Reilly, 2010.

[5] Castledine, E.; Eftos, M.; Wheeler, M.: *Vytváříme mobilní web a aplikace pro chytré telefony a tablety*. Brno: Computer Press, první vydání, 2013.

[6] Green, I.: *Web Workers*. Sebastopol: O'Reilly Media, 2012.

[7] Kadlec, T.: *Responzivní design*. Brno: Zoner Press, 2014.

[8] King, A. B.: *Zrychlete své www stránky*. Zoner Press: Brno, vyd. 1. vydání, 2004.

[9] Lewis, P.; Irish, P.: High Performance Animations. 2013.
URL http:
//www.html5rocks.com/en/tutorials/speed/high-performance-animations/

[10] Marcote, E.: Responsive Web Design. *A List Apart*, , č. 306, 2010.
URL http://alistapart.com/article/responsive-web-design

[11] Miller, R. B.: Response time in man-computer conversational transactions. *Proceedings of the December 9-11, 1968, fall joint computer conference, part I on - AFIPS '68 (Fall, part I)*, 1968: str. 267.
URL http://portal.acm.org/citation.cfm?doid=1476589.1476628

[12] Vujovic, M.: CSS animations and transitions performance: looking inside the browser. 2014.
URL https://blogs.adobe.com/webplatform/2014/03/18/
css-animations-and-transitions-performance/

Príloha A

Obsah CD

Technická správa

Technická správa vo formáte *pdf* sa nachádza v adresári *technicka_sprava*. Zdrojové kódy pre LaTeX sa nachádzajú v adresári *technicka_sprava/tex*. Technickú správu je možné vygenerovať pomocou príkazu *make*.

Demonštračná aplikácia

Zdrojové súbory k demonštračnej aplikácií sa nachádzajú v adresári *demo_app/src*. V adresári *demo_app/src/javascript* je možné príkazom *make* s príslušnými parametrami, popísanými v prílohe B, preložiť javascript aplikácie. HTML, CSS a ďalšie súbory sa nachádzajú v adresári *demo_app/src/www*. Do tohto adresáru sa prekladá i javascript.

Preložená aplikácia sa nachádza v adresári *demo_app/bin*. V jednotlivých adresároch *ios*, *android*, *winphone* sú inštalačné súbory jednotlivých platforiem a v adresári *browser* je aplikácia dostupná pre spustenie v bežnom internetovom prehliadači spustením *index.html*.

Video a plagát pre demonštráciu projektu sa nachádzajú v adresároch *video* a *plagát* v koreňovom adresári CD.

Príloha B

Inštalácia a spustenie

Táto príloha popisuje možnosti prekladu, inštalácie a spustenia demonštračnej aplikácie. Všetky potrebné súbory súbory sú uložené na CD v zložke *omo-mobile*.

Preklad

Pre preklad aplikácie je potrebné v adresári *momo-mobile/javascript* preložiť príkazom ***make***. Príkaz make má rôzne parametre prekladu:

- **make all** alebo len **make** - zapne ladenie javascriptu na adresu localhost:9810, pre spustenie aplikácie je nutné otvoriť v prehliadači súbor *omo-mobile/javascript/src/index.html*.
- **make build** - v adresári *omo-mobile/www* vytvorí preloženú lokálne spustitelnú verziu aplikácie. Je možné ju otvoriť v prehliadači otvorením súboru *omo-mobile/www/index.html*.
- **make remote-build** - aplikácia sa preloží a nahrá na cloudovú službu build.phonegap.com, následne bude dostupná na adrese zakódovanej v QR kóde na obrázku B.1 Pre túto voľbu je potrebné mať nainštalovaný PhoneGap[1].
- **make sim-[*ios, android*]** - Aplikácia sa spustí v simulátore príslušnej platformy. Je nutné mať nainštalovaný príslušný simulátor pre Android či iOS a PhoneGap [2].
- **make [*remote-build-, build-, sim-*]fps-[*with, without*]-hw-acc** - zapne zobrazovanie FPS s podporou spustenia testov (6.3). Prefixy build alebo sim majú rovnaký význam ako u predchádzajúcich príkladov. *with/without* rozhoduje o preklade s hardvérovými akceleráciami alebo bez.
- **make [*remote-build-, build-, sim-*]worker-test[*-with-postprocessing*]** - zapne zobrazovanie aplikačného času s možnosťou spustenia testu webworkerov (6.3).

Inštalácia na zariadenie a spustenie

Aplikáciu je možné spustiť po preklade z príslušného adresáru (viď. sekcia Preklad) spustením súboru *index.html*. V prípade nefunkčnosti je nutné v prehliadači povoliť CORS (cross-origin-requests). Alebo nahrať adresár *www* na reálny alebo virtuálny server. Pri zosnímaní kódu na obrázku B.1, alebo z adresy „link.klokantech.com/omom“ sa aplikácia automaticky

[1] http://phonegap.com/install/
[2] http://docs.phonegap.com/en/edge/guide_platforms_index.md.html

Obrázek B.1: QR kód stačí pre automatickú inštaláciu aplikácie zosnímať do mobilného zariadenia.

stiahne a začne inštalovať. Podporované platformy sú iOS a Android. Windows Phone je netestovaný. iOS funguje len na jailbreaknutých alebo vývojárskych iOS zariadeniach z dôvodu certifikátov. Android bez obmedzenia. Aplikácia je primárne vyvíjaná a testovaná na systéme iOS. V prípade nainštalovaného emulátoru, je možné aplikáciu spustiť príkazom make (viď. sekcia Preklad).

Testované zariadenia

Aplikácia bola testovaná a jej funkčnosť je odladená pre zariadenia Apple iPad 2+, iPhone 4+, Google Nexus 5 (Android 4.4) a internetové prehliadače Google Chrome 36 a Safari 7.

Príloha C

Vlastnosti a nastavenia pre natívny vzhľad webovej aplikácie

Nachádza sa tu výčet jednoduchých nastavení a trikov ako prevažne vizuálne priblížiť webovú aplikáciu natívnej. Tipy pre dosiahnutie viac natívneho vzhľadu webovej aplikácie[1]:

U každého typu je uvedený spôsob ako to zabezpečiť v bežnej stránke alebo akú vlastnosť zapísať do PhoneGap[2] config.xml súboru:

Zamedzenie zväčšovaniu a zmenšovaniu viditelného obsahu stránky

HTML header:

```
<meta name="viewport" content="width=device-width; initial-scale=1.0;
    maximum-scale=1.0; user-scalable=0;"/>
```

Phonegap config.xml:

```
<preference name="EnableViewportScale" value="false"/>
```

Počiatočné nastavenie je zakázané.

Nastavenie zobrazenia na celú obrazovku

HTML header:

```
<meta name="apple-mobile-web-app-capable" content="yes" /> //len iOS
```

Phonegap config.xml:

```
<preference name="Fullscreen" value="true" />
```

Štýl stavového panelu (iOS)

HTML header:

[1] Tieto informácie sú zozbierané zo zdrojov: http://docs.build.phonegap.com/en_US/3.3.0/index.html, http://www.marcofolio.net/webdesign/build_native-looking_apps_for_ios.html, http://www.mikedellanoce.com/2012/09/10-tips-for-getting-that-native-ios.html

[2] Popisovaná je verzia PhoneGap 3.3.0

```
<meta name="apple-mobile-web-app-status-bar-style" content="black-translucent" />
// black-trasculent - aplikácia je na celej obrazovke čiastočne zakrytá stavovým panelom
```

Phonegap config.xml:

```
<preference name="ios-statusbarstyle" value="black-opaque" />
//black-opaque - čierny status bar - black-translucent je tiež podporovaný ale v˜rámci phonegap je zhodný s˜black-opaque, základné nastavenie je default - šedý panel
```

Nastavenie ikony

HTML header:

```
<link rel="apple-touch-icon" href="images/apple-touch-icon.png" /> //iOS
//PNG súbor a rozlíšenie musí byť 114x114 pixelov
```

Phonegap config.xml: Android príklad:

```
<icon src="xxhdpi.png" gap:platform="android" gap:qualifier="xxhdpi" />
```

iOS príklad:

```
<icon src="icon-50@2x.png" gap:platform="ios" width="100" height="100" />
// je možné zadefinovať ikony pre rôzne zariadenia, platformy a rozlíšenia
```

Nastavenie štartovacej obrazovky

HTML header:

```
<link rel="apple-touch-startup-image" href="iPhonePortrait.png" /> (iOS)
<link rel="apple-touch-startup-image" sizes="768x1004" href="iPadPortait.png" /> (iOS)
// PNG súbor, 320x460 pixelov pre iPhone/iPod a 1004x768 pixelov pre iPad
```

Phonegap config.xml:

```
<gap:splash src="splash/ios/Default-568h@2x˜iphone.png" gap:platform="ios" width="320" height="480" />
//sú podporované i ostatné platformy
```

Viac informácií o phonegap parametroch je možné nájsť na stránkach PhoneGapu[3].

Vypnutie Rubber Banding - elastické rolovanie

Phonegap config.xml:

```
<preference name="webviewbounce" value="false" />
```

Nastavenie orientácie obrazovky - zakázať rotáciu

Phonegap config.xml:

```
<preference name="orientation" value="portrait" />
```

[3] http://docs.build.phonegap.com/en_US/3.3.0/index.html (kapitola Configuring)

Príloha D

Trieda pre JSONP vo Web Workeri

Ukážka ako vytvoriť triedu s podporou JSONP aby fungovala v rámci Web Workerov. Trieda založená na *goog.net.Jsonp* z Google Closure.

```
1  goog.provide('omo.net.Jsonp');
2
3  goog.require('goog.net.Jsonp');
4  goog.require('omo.Const');
5
6  /**
7   * Creates a new cross domain channel that sends data to the specified
8   * host URL. By default, if no reply arrives within 5s, the channel
9   * assumes the call failed to complete successfully.
10  *
11  * @param {goog.Uri|string} uri The Uri of the server side code that
        receives
12  *     data posted through this channel (e.g.,
13  *     "http://maps.google.com/maps/geo").
14  *
15  * @param {string=} opt_callbackParamName The parameter name that is used to
16  *     specify the callback. Defaults to "callback".
17  *
18  * @constructor
19  * @extends {goog.net.Jsonp}
20  * @final
21  */
22 omo.net.Jsonp = function(uri, opt_callbackParamName) {
23   goog.base(this, uri, opt_callbackParamName);
24 };
25 goog.inherits(omo.net.Jsonp, goog.net.Jsonp);
26
27
28 /**
29  * Sends the given payload to the URL specified at the construction
30  * time. The reply is delivered to the given replyCallback. If the
31  * errorCallback is specified and the reply does not arrive within the
32  * timeout period set on this channel, the errorCallback is invoked
33  * with the original payload.
34  *
35  * If no reply callback is specified, then the response is expected to
36  * consist of calls to globally registered functions. No &callback=
37  * URL parameter will be sent in the request, and the script element
38  * will be cleaned up after the timeout.
39  *
```

```
 * @param {Object=} opt_payload Name-value pairs.  If given, these will be
 *     added as parameters to the supplied URI as GET parameters to the
 *     given server URI.
 *
 * @param {Function=} opt_replyCallback A~function expecting one
 *     argument, called when the reply arrives, with the response data.
 *
 * @param {Function=} opt_errorCallback A~function expecting one
 *     argument, called on timeout, with the payload (if given), otherwise
 *     null.
 *
 * @param {string=} opt_callbackParamValue Value to be used as the
 *     parameter value for the callback parameter (callbackParamName).
 *     To be used when the value needs to be fixed by the client for a
 *     particular request, to make use of the cached responses for the
    request.
 *     NOTE: If multiple requests are made with the same
 *     opt_callbackParamValue, only the last call will work whenever the
 *     response comes back.
 *
 * @return {!Object} A~request descriptor that may be used to cancel this
 *     transmission, or null, if the message may not be cancelled.
 * @override
 */
omo.net.Jsonp.prototype.send = function(opt_payload,
    opt_replyCallback,
    opt_errorCallback,
    opt_callbackParamValue) {
  var payload = opt_payload || null;

  var id = opt_callbackParamValue ||
      '_' + (goog.net.Jsonp.scriptCounter_++).toString(36) +
      goog.now().toString(36);

  if (!goog.global[goog.net.Jsonp.CALLBACKS]) {
    goog.global[goog.net.Jsonp.CALLBACKS] = {};
  }

  // Create a new Uri object onto which this payload will be added
  var uri = this.uri_.clone();
  if (payload) {
    goog.net.Jsonp.addPayloadToUri_(payload, uri);
  }

  if (opt_replyCallback) {
    var reply = goog.net.Jsonp.newReplyHandler_(id, opt_replyCallback);
    goog.global[goog.net.Jsonp.CALLBACKS][id] = reply;

    uri.setParameterValues(this.callbackParamName_,
                           goog.net.Jsonp.CALLBACKS + '.' + id);
  }

  // TODO: Error catching here!!!
  self.importScripts(uri.toString());

  /*var deferred = goog.net.jsloader.load(uri.toString(),
      {timeout: this.timeout_, cleanupWhenDone: true});
  var error = goog.net.Jsonp.newErrorHandler_(id, payload, opt_errorCallback
      );
```

```
97    deferred.addErrback(error);*/
98
99    return {id_: id};
100 };
```

Príloha E

Plagát